University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: **Computer Science**	Spring semester, 2018 Open / Restricted access
Writer: **Anju Alex**	…………………………………………… (Writer's signature)
Faculty supervisor: **Dr. Tomasz Wiktorski**	
Title of thesis: **Machine Learning Based System Health Check Analyzer For Energy Components**	
Credits (ECTS): **30**	
Key words: Supervised Machine Learning , Elastic Search , Logstash , Log analysis	Pages: …62………… Stavanger, Date/year: 15/06/2018

Machine Learning Based System Health Check Analyzer for Energy Components.

Master’s thesis

Anju Alex

Faculty of Science and Technology
University of Stavanger
Norway

June 2018

Acknowledgements

I would like to express my sincere gratitude to Dr. Tomasz Wiktorski, my supervisor, for his support and guidance throughout the thesis. Your valuable comments and suggestions helped me to be on track with the thesis without losing focus.

Furthermore, I would like to say a big thank you to Egil Ølberg, my mentor at Tieto A.S, for engaging me in such an interesting topic for the thesis, for bringing in different ideas and being very supportive. I would also like to thank Tieto A.S for providing good support to do the thesis and giving access to the server logs and EC application.

Special thanks go to my friends Martin and Waqas with whom I have shared my difficulties, ideas, thoughts and moments of victories during the thesis. Their inspiring words kept me going. I am also very thankful to my Mother who was here for helping me during the thesis days. Without her support it would have been difficult to carry on.

Last but not least, I would like to thank my Husband Alex Anthony and my 3 kids – Alicia, Carol and Olivia for being my pillars of strength and for just showing immense confidence in me no matter what. Their belief in me and constant support always made me go one step further from where I was.

Anju Alex

University of Stavanger

Abstract

In any system health check is an important measure, which provides details on how the system is performing and whether there is a need for an intervention manual or automated to correct any anomaly. There are several approaches to measure the system's health, server logs being the one used for this thesis.

In this thesis a prototype of a health check analyzer tool is developed for a product called Energy components. This health check analyzer tool can be used to monitor the system state based on the generated server log files.

In this study supervised machine learning techniques have been applied to do automated log analysis. Incoming logs are read by Logstash, which filters them and extracts useful information and stores them in Elasticsearch. Using Elasticsearch, the parsed structured log files are indexed, which is then read by the machine learning model. Features from the contents of the logs are extracted using different vectorizers and further used to train machine learning model. Several variants of text classification algorithms are experimented and compared, in order to select the most suitable model for the problem being addressed in this study. K fold Cross validation and F1-score, performance matrix and learning curve are used to evaluate different learning models. A high accuracy rate of 94% with 93% precision and 0.058% standard deviation is achieved by using different machine learning algorithms and by varying the tuning parameters. The case study results showed that Support Vector Machine algorithm with hashing vectorizer gave the best accuracy results among the other compared algorithms.

Table of Contents

Index of Figures

Index of Lists

Index of Tables

Chapter 1 - Introduction

Collecting feedback about the state of any medium sized or large system can be a very daunting task. Hence, it is a normal programming practice to log all the major events happening during the system operation, be it good or bad, in some form of log files. These logs can then further be analyzed to detect the anomalies, assess the system state or find the cause of an anomaly etc. However due to the growing complexity of today's IT systems, the log files also grow in size tremendously, thereby making it difficult to analyze them manually for error detection. In such a big data scenario, using machine learning techniques to do automated log analysis becomes interesting and desirable.

Using Machine learning models, log files could be analyzed in real-time to find unusual /suspicious behaviors for e.g. mysterious system crashing or data or memory leaks etc. Using the result from these models user could be alerted so that corrective actions can be performed before it is too late and cause serious financial loss.

Energy Components (EC) is a hydrocarbon accounting product that accounts for hydrocarbons, from the point they are drilled from wells to the point they are sold out. Because of its real time nature and the fact that it deals with highly expensive hydrocarbons, it becomes increasingly important that the health of an EC system is monitored properly and preventive measures are taken on time should there be a need.

This project aims to extract effective features from the free text contents of server logs generated by EC, detect the abnormal/faulty logs automatically through machine learning techniques and also investigate the effectiveness and efficiency of different algorithms for this prototype, by comparing the results. Further this projects aims to provide a good alerting system that can alert the user if a faulty log is found.

Background and Motivation

Tieto A.S has been involved in providing solutions for Oil and Gas industry since 1985. Tieto's market leading product Energy Components (EC) helps its users to manage their hydrocarbons and business processes in all the different areas like production, transport, sales and revenue. Energy Components has over the years become an industry standard for many of the Oil giants all over the world [1].

In a complex application, like Energy Components, multiple developers work together to change the software to improve the product continuously by adding new features and extending the existing ones. Generally, the testers are not as familiar with all the code level system details as the developers. Moreover, even the system developers cannot foresee all problems. Chances are that there can occur unknown, new problem due to growing and continuously integrated and updated system. It is very important to find these errors when the system is in production, which otherwise will result in huge financial loss for the user and big questions against the products quality for Tieto, which in turn will affect Tieto's reputation as provider of global leading hydrocarbon accounting software solution.

Most of these errors can be caught by analyzing the server logs generated by the system. Manual log analysis is a daunting task for large systems. In reality, manual log analysis plays a vital role when automated testing/manual testing cannot find out all the problems with the system as many of the errors and warnings are not seen in frontend. They occur in the backend and are direct evidence of system's ill health and these errors and warnings are logged in the server log. However, in such a complex system analyzing server logs usually involves a wide variety of techniques which requires multiple experts of different fields to come together.

With the Continuous Integration, Business process modelling etc. in place in EC, the system dumps huge server logs every day and most of them are not interesting. It is expensive and is time consuming to train a lot of experts to detect all interesting logs correctly and manually analyze them. Therefore, this work is mostly done by testers and mostly these testers are not system level experts and they select few suspicious looking log files instead of looking into all generated logs and use simple tools or commands (e.g.: find, grep) to search some keywords (e.g. abort, connection failed, error, fatal) which may help to detect problems based on their previous personal experiences.

However, analyzing logs is more difficult than this. Since a small change in configuration can lead to a completely different log message, logs can look ambiguous quiet soon for a non-system expert. In order to identify problems through analyzing logs, the tester must understand the full behaviors of the system and the logs it produces. But due to the frequently changing code base, and new features added to different modules in EC and continuous improvement of EC framework, it is difficult to find such an expert who knows all what is needed.

All these problems make automated log analysis very desirable. Also one another main motivation for this thesis is that EC will soon be implemented in cloud. Hence it becomes even more critical to get alerted if there is any anomaly as any delay could implicate huge monetary loss. This arises the need for automated tools for health check analysis of the system that require minimal or no human assistance and could alert the user on time if an anomaly is detected.

Below listed is the main thought process behind choosing machine learning approach for Tieto (EC):

- **Explicit programming to analyze logs is hard**

EC system is usually composed of various parts and those parts are developed in different programming languages. Hence the contents of different component’s log can be greatly different. There is no specific structure standard for EC log files, hence it is not so straightforward to develop a specific program do the log analysis by parsing logs and interpreting its content. Even if a specific log analyzer program is developed using a specific software, it cannot be used without frequently updating the logic as EC is constantly updated and upgraded.

Additionally, as mentioned previously, even for human beings, it is hard to identify problems from tons of log data correctly, and hence it is next to impossible to write a perfect log analyzer program by a programmer.

- **Machine learning will help**

Since training many resources to become experts in all modules in EC is time taking and expensive, it might be a good idea to consider to train a program to learn by itself, which is precisely what is done by machine learning techniques. Machine learning models can efficiently work with large scale data by gathering experiences from learning features and patterns on training datasets and apply them on a new datasets which could be different from the training data. Many machine learning techniques have been successfully used in solving complex problems, ranging from learning to detect fraudulent credit card transactions, learning web user's browsing preferences to learning of driverless vehicles to drive on public roads [12].

Machine learning algorithms can thereby automate the log analysis process in a much efficient manner in a much shorter time than humans. Its main benefit is that it does not need to know all the truth about all the problems beforehand. Also, it is not a necessary condition that the developer of a machine learning system has any expert knowledge in log analysis but is a plus in finding effective features [2].

Thesis Goal and Outline

The primary objective of this master's thesis is to develop a prototype for machine learning based system health check analyzer which is based on the concept of automatic detection of suspicious logs through log analysis. It needs to be able to parse an incoming log, extract useful information from it and then be able to classify it as erroneous or normal log and needs to be able to alert the user in case of finding an erroneous log. It needs to be very efficient in terms of dealing with big data considering the huge size of log files generated by the system. This health check analyzer can at any point, based on the server log, should be able to give a quick overview of system state.

With the above arguments in mind, the following goals are expected to achieve:

- Set up Elasticsearch, Logstash and Kibana (ELK stack) on a dedicated Virtual Machine.
- Set up Energy Components Application server using wildfly-10.1.0 in an internal Tieto machine.
- Generation of different scenario based log files from Energy Components for training and testing.
- Configure Logstash to consume logging messages from log4j.
- Log parsing, preprocessing and reduction of log files inside Logstash.
- Configure Elasticsearch to receive and store the preprocessed log files from Logstash.
- Read the indexed log files from Elasticsearch through a python program. Perform labelling of the read files as normal or erroneous.
- Split the log files into test and train sets.
- Perform feature extraction in the labelled training dataset. Train the machine learning model using Training set.
- Test the model for prediction using the test dataset and measure the accuracy and precision of the predictions. Also perform k-fold cross validation on the model using all the data and find the score.
- Configure an alerting mechanism which sends an email notification if an erroneous log is detected.

Workflow

This section describes the general workflow of the prototype created for this thesis. Figure 1 gives an overview of the steps involved in obtaining the objectives of this thesis.

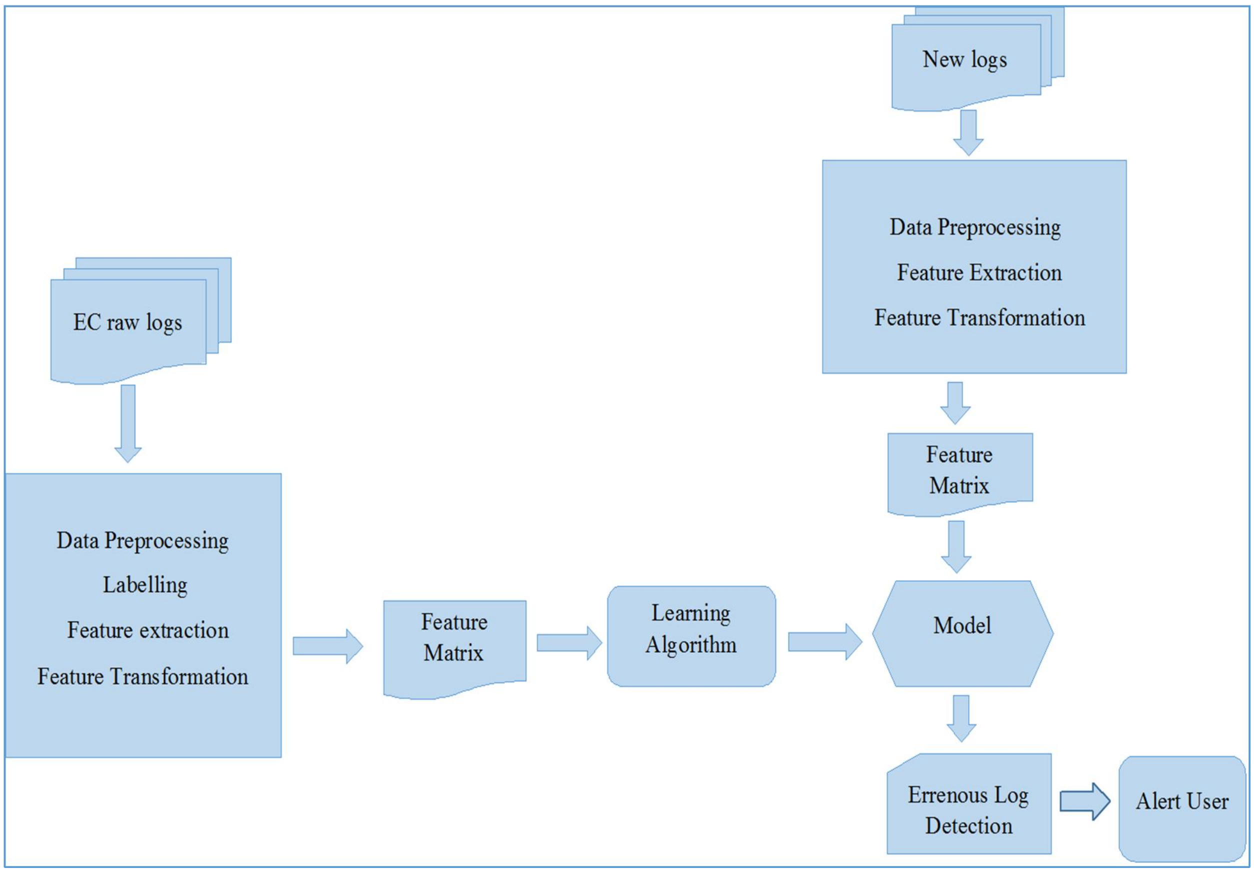


Figure 1. Work flow of Health check analyzer

Thesis Structure

The rest of this thesis report is structured as follows:

Chapter 2 – Theoretical background for the thesis is presented.

Chapter 3 – Framework of the prototype is presented.

Chapter 4 – Methodology followed by this thesis is explained.

Chapter 5 – Some case studies and their results are presented. Many of the factors affecting the learning results are compared and result of the best working algorithm is presented. Finally, conclusion of this thesis work is presented along with possible future work in this area.

Chapter 2 - Theoretical Background

This chapter describes the theoretical foundation of this work based on existing literature. Below described are the core concepts used in creating this prototype.

Log Mining

With the advancement of science and technology, there is tons of digital data produced every hour by systems around the world. A large chunk of such digital data consists of log files. Data mining is the science of obtaining useful information from such huge digital data repositories [8].

Extracting right and useful information from large amount of data can be done in numerous ways, and it is vital to ask the right questions in order to collect information which is useful [9]. Two desirable features of a data mining algorithm are that it must be generic to the type of data format coming in and that it can normalize the incoming data. Also it is highly desirable that data mining algorithms are efficient in terms of time and resources and are scalable as they generally deal with huge datasets.

A subfield of data mining is log mining, which is the art of collecting important information from large logs or log repositories. The reason why log mining is important is described in Background and motivation section of this report (See Chapter 1 section 1). As in data mining, log mining also faces similar kind of challenges like too little or too much data coming in, that format of incoming log can be different, and that there could be duplicate log files coming in. Apart from this another challenge is to know which part of log is interesting and which part to ignore. One approach is to take out expected events/messages, so that only unexpected or not normal data can be analyzed. Depending on the log data, interesting could be defined as rare events, suspicious or not normal events, and large counts of uninteresting things or strange combinations [9].

The very first step to mine log data is to define specific goals as to which questions are interesting to the user and should be asked and can the resulting information be used to make a good solution. Thereafter, log data has to be collected, pre-processed and cleaned, possibly reduced. After this one of the suitable methods (Supervised learning in this case) has to be chosen based on the dataset and the goals (See Chapter 1 section 2).

Once learning model has been applied, the results must be interpreted and possibly visually presented to analyze [9]. Also an important thing to keep note of is that log reduction must be applied after log collection and not before or during the log collection. This is because many presumptions on data can lead to preventing new/unexpected discoveries on data.

Log mining approaches

Log mining can be done by using relatively straightforward statistical methods such as measurement of frequency or mean. Standard deviation probability or standard error could also be used for log mining purposes. Also it can be interesting to look at if any distribution effects the dataset. Normally, it is needed to establish a baseline for the data, based on past data. When baseline is established, it is important to age out old data after a suitable amount of time [13].

Another age old technique that can be applied to the dataset is string/literal matching or regular expressions [9]. This can be however be extended to discover pattern based similarity between strings [20].

Yet another approach is to divide strings into tokens, which can be describes as a sequence of letters split by a white space. It can be said that a token to some extent corresponds to a word in a sentence. However, some tokenizers also take into account punctuations as well even though it is not a part of the word. The breaking down of text to tokens can help in understanding of the composition of the text by for example monitoring the frequency of the tokens [26].

One of the important aspect of data mining is if it is applied on real time data or on historical data. If the model is based on historical data, there could be need for it to be updated continuously. Another solution is to build the model using real time logs, which is described as data stream mining [21]. This however comes with its own set of challenges on how to apply mining techniques on a rapid, free flowing and large set of data.

Big data

Although no one can reach to a good consensus on what is Big data, most will agree that bulk of server log files generated by a large scale application can be considered as big data. This is because of the huge load of information routinely generated and placed into these logs [23]. One possible way big data can be defined is as data whose sheer volume or size forces us to look beyond the traditional data handling methods, which stresses on the fact that as volume of data increases , data handling capabilities need to be increased [22].

As the size of data increases, mere storage and efficient analysis of this becomes challenging. Since for this project large volume of log data has been analyzed, big data analytics has been of much importance for the study. Hence in this project Elasticsearch, an analytic search engine specially made for big data handling, has been used.

Machine learning

Machine Learning is the science of making computer systems learn through experiences, without being programmed explicitly on what to do when a task is being performed. Here the main focus is to design a system that can learn just like humans from experiences or examples through methods like training or recognizing patterns in data etc. Most of the machine learning algorithms are designed to learn and improve as it is exposed to more and more data.

Figure 2 shows the different types of machine learning methods and their usage in real world. As seen in the figure Machine learning can be broadly classified as supervised learning and unsupervised learning depending on if there is training data available or not.

Here in this thesis supervised machine learning algorithm has been used, where a model is trained using a labelled training data set so that it learns the features of training dataset corresponding to the labels of the dataset. When new input data is fed to ML algorithm, it makes a classification on the basis of the learning it has done previously.

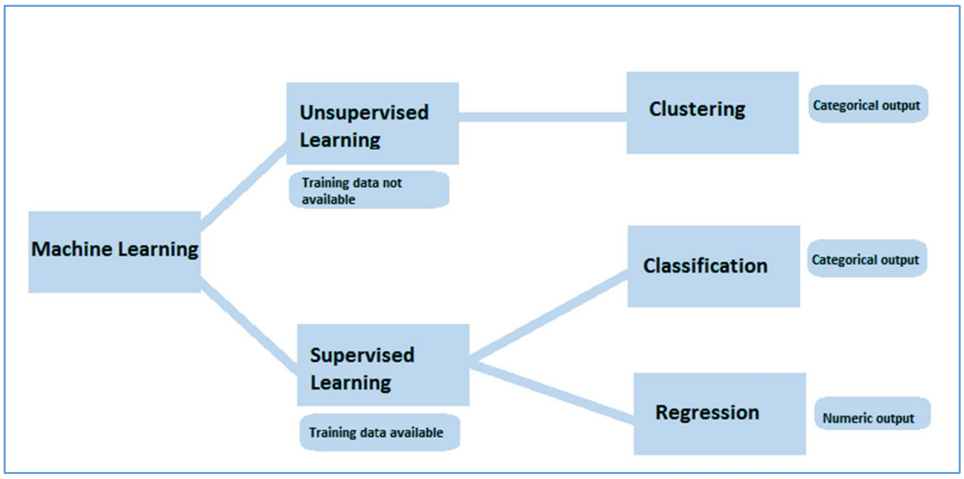


Figure 2. Machine Learning Diagram

Supervised learning paradigm

Supervised learning is the method in which the algorithm learns from examples or instances provided to it. Supervised learning can be defined as the machine learning task of learning a function f(X) that maps an input X to an output Y based on example input-output pairs or training dataset [14]. It derives a function f(X) from the labelled input training data to arrive at Y.

Here each training sample is a pair (x,y) comprising of an input value and a preferred output value. The learning algorithm examines the training sample and tries to reproduce the underlying function, which can also be called as a mapping function. This mapping function can then be used for mapping new input samples. In an ideal scenario, the algorithm will correctly classify unseen samples. This necessitates the supervised learning algorithm to use the mapping function and generalize from the training examples to unseen instances using statistical or other mathematical ways.This can be expressed by the following formula:

$$Y = f(X) \quad , \qquad [29]$$

where X is the input variables,

Y is the output variable and

f(X) is the underlying mapping function.

Mathematically speaking, here we have an input variables (X) and an output variable (Y) and we use an algorithm to learn the underlying mapping function from the input to the output.

The supervised machine learning algorithm generates a self-improving function g which maps the same data X to an output Y ′ expressed by the formula below:

$$Y' = g(X) \quad ,$$ [29]

where g(X) is the self-improving function.

The objective is to make function g to be as close to function the true function f as possible. How good or bad the current version of g is doing can be derived by comparing its output Y ′ to the true output Y given by the training examples.

For finding erroneous logs, this kind of learning algorithm needs a pre-classified (pre-labelled) set of training data that outlines normal logs and abnormal logs. Supervised algorithms may not be able to properly handle data from an unexpected region, as it was not covered in the training data [15]. Hence it is essential for the training data to cover as much of normal behaviors as possible, including examples of normal and erroneous data. To perform this, all the features of data that can be important for describing the dataset needs to be carefully chosen and included in training [30].

Supervised learning problems can broadly be classified into regression and classification problems.

1. Classification: A classification problem is a problem in which one needs to classify an incoming data sample into a category like category ‘A’ or category ‘B’. Here the output variable is discreet like a class label.

2. Regression: A regression problem is a problem in which one needs to predict a continuous output variable, where the output variable is a real value like an integer or float.

Feature Extraction from Logs – Related works

EC server logs include a lot of information in them which include both numerical data and non-numerical data. For some kinds of analysis, these numerical data (e.g. memory, CPU load etc.) is extremely important as features. But for other type of log analysis, the meaningful non-numerical data can be very interesting like for e.g. some keywords in the log messages can indicate the current system state. The features reflecting the text contents of the logs are the focus of this thesis.

In the area of log analysis, there is no commonly recognized standard for feature selection. Although this area is still under research, various methods have been tried and some illustrative features from logs have been studied and documented.

Since logs can be regarded as textual data, natural language processing (NLP) techniques can be applied on them. NLP is a branch of computer science and artificial intelligence which engages in automated manipulation or processing of the content of a text or speech (termed as natural language) using a software[33]. It provides a tool for interaction between humans and computer.

Many NLP and text classification studies focus on extracting N-gram frequencies [24] from the logs. An n-gram is a continuous sequence of n items from a given sample of text or speech [18]. It is widely used when doing text categorization. There are studies on language classification and subject classification of newsgroup articles based on N-gram frequencies. For example N-gram-based text categorization [24] is one such study. In the study - Anomaly Detection from Network Logs Using Diffusion Maps [34], N-gram frequencies of network logs are used as features to perform intrusion detection. It is a common practice to take a whole word instead of a letter as a gram and also generally contiguous sequences of n words are used rather than using permutation of all words.

TF-IDF [14] which stands for term frequency – inverse document frequency is another popular method used in natural language processing area. In TF-IDF, frequency of one word in a specific document in considered and is balanced by the frequency of words in the corpus. In the study - Detecting Large-Scale System Problems by Mining Console Logs [10], this measure is used in the message count feature and it has improved the accuracy and precision in detection of abnormal logs. Many times the combination of TF-IDF and n-gram is used together as features. Since in this current thesis TF-IDF has been used, there is a dedicated section in this document to explain this method in detail.

In addition, many researches use data mining techniques to mine patterns from the data and use them as features. Also there are some log analysis researchers that extract structured information from logs by plain programming. Their study is based on the thinking that although logs are different, they follow a basic underlying template and hence it is possible to define specific important attributes for each logs and extract them as features.

Text feature extraction

Machine Learning algorithms are widely being used for analyzing text. But raw data which is a mere collection of a series of symbols cannot be directly fed to the learning algorithm as most of these algorithms expect not just text of any length but numerical feature vectors of fixed size. The process of extracting numerical feature vectors from a raw text document is called vectorization. In order to extract useful numerical feature vectors from raw text, following steps can be done in scikit learn (a python tool used to create learning model for this thesis):

- Tokenizing – by using token separators (mostly punctuations or blank spaces), the text file is split to tokens.
- Counting- the number of times each token has occurred in a document is counted.
- Normalizing and weighting – the count is normalized by weakening importance of the tokens that occur in the majority of samples / documents in the corpus.

This specific method which involves tokenization, counting and normalization is called the Bag of Words representation [35]. In this method a feature would be each individual token's occurrence count and multivariate sample is the vector of all the token frequencies for a given text file [35]. Thus a whole corpus of documents can be represented by a matrix with one row per text document and one column per token or word appearing in the corpus [35].

Tf–Idf term weighting

In every big English document corpus some words are very common for e.g. ‘a’, ‘an’, ‘is’, ‘this’, ‘the’ etc. These words are mostly connectors and do not really express much information about real content of a document. Hence is all the words of a document are directly fed to the learning algorithm, these connector words can adversely affect the learning and classification as these can shadow the frequencies of more important and interesting terms which could be occurring not as frequent as these terms [35].

Hence a method called tf-idf is used for term weighting of each token. Tf is the term frequency while Idf denotes inverse document frequency.

The term frequency or tf (t,d) is the count of the number of occurrences of a term in a document.

The Inverse document frequency, Idf is computed as:

$$idf\ (t) = \log\frac{1+n_d}{1+df\,(d,t)} + 1 \quad , \qquad [35]$$

where n_d is the total number of documents in the corpus,

d denotes a document,

and d*f* (*d*,*t*) is the total number of documents that have the term *t*.

Tf-idf is computed by multiplying tf (t,d) by idf (t) as shown below:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t) \ , \qquad [35]$$

The tf-idf vectors resulting from this are then normalized by the Euclidean norm given below:

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2+v_2^2+\cdots v_n^2}} , \qquad [35]$$

where v is a vector in the feature space consisting of vectors $v_1, v_2, \ldots\ldots v_n$.

TF-idf was earlier used as a weighing scheme for ranking pages in search engines, but now it has been found to be very effective when used for classification and clustering.

Text classification

Text classification is the technique of smartly classifying text into categories. Text classification can be done manually or by explicit programming or by machine learning. When machine learning is used to automate text classification, the whole process becomes efficient and very fast.

Text classifier can function on variety of textual datasets. In this kind of classification the text of the dataset is analyzed to figure out the intent or overall ‘summary’ of the textual data. One can choose to train the model with labelled data or work with the raw unstructured text data depending on the type of classifier used (supervised or unsupervised).There are various applications using both these categories of text classification.In this study supervised text classification is the method used and is elaborated in the below section.

Supervised Text Classification

Supervised classification of textual data is performed when some class categories have already been defined. It is based on training and testing theory. Pre-tagged data has to be fed to the learning algorithm to train it. The algorithm is trained on this fed labeled dataset and would give the correct output (one of the predefined class categories). During the testing phase, previously unseen data is fed to the algorithm. Based on what it has learned during the training phase, algorithm classifies the new data into categories. This is also called as predictive text mining.

A supervised text classification task comprises of two steps. The first step is the training process. A fixed set of n documents $D = \{d_1, d_2, d_3, \ldots d_n\}$ are labeled and used as the training set $\{(d_1, l_1), (d_2, l_2),(d_3,l_3), \ldots , (d_n, l_n)$. This is how it is specified to learning algorithm that document d_1 belongs to category l_1, d_2 to l_2 etc.Each document d_i is defined as a feature vector in the feature space:

$$F = \{f_1, f_2, f_3 \ldots f_m\}. \qquad [38]$$

The algorithm thus correlates between features and label.

The second step in the process is testing or prediction. When presented with an unlabeled set of documents, the classifier predicts the labels of each of them.

Stemming

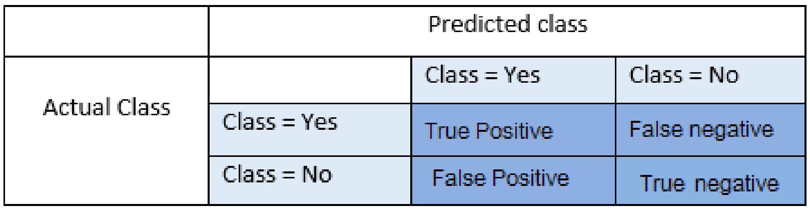
Stemming is a technique which is used to find out the root of a word. For example, the words, connecting, connection, connector, connects all can be stemmed to the word ‘CONNECT’. Stemming changes words to their roots, which requires much linguistic knowledge of the language. The reasoning for doing stemming is that words with same word root typically describe same or somewhat close concepts in the document and so such words can be reduced to one word (the stem) by using stemming. In this study, snowball stemming algorithm is used. Snowball algorithm uses a string processing language called snowball for performing stemming and hence the name.

Stop-word Removal

Stop-words are typical functional words in any language, which occur very frequently in text but do not carry any information like pronouns, conjunctions, propositions, auxiliary verbs etc. [36]. If the case of English language is taken, it can be seen that there are more than 400 stop-words. Some examples of stop-words can be ‘to’, ‘the’ etc. Although these words are very important for the grammar, these normally are of no use when analyzing text. Hence in this study stop-words removal also has been applied to the text in few case studies. This is done in scikit learn by passing an argument (stop words='english') to the count vectorizer function.

Confusion Matrix

A confusion matrix is an extract of prediction results for a classification model [37]. It shows how much a classifier is confused while making a prediction. This is a good measure to evaluate the performance of the classifier. Figure 3 shows an example of a confusion matrix.



	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False negative
	Class = No	False Positive	True negative

Figure 3[37]. Confusion Matrix

The main terms in a confusion matrix are:

- True Positives (TP) – This is the count of rightly predicted positive values.
- True Negatives (TN) - This is the count of rightly predicted negative values.
- False Positives (FP) – This is the count of wrongly predicted positive value which happens when real class is ‘no’ and prediction is made towards ‘yes’ class.
- False Negatives (FN) – This is the count of wrongly predicted negative value which happens when real class is ‘yes’ but prediction is made towards ‘no’ class [38].

Accuracy

Accuracy is a performance measure of the classification model and it is merely the ratio of rightly predicted values to the total values. It is natural to think that higher the accuracy, better the model. This is correct when the dataset is almost symmetrical where FP and FN are nearly same [38]. Hence, other measures must also be taken into consideration to evaluate the performance of the model. For the current study an accuracy of 0.93 has been achieved which implies the model is approx. 93% accurate when the learning algorithm used is SVM algorithm.

$$\text{Accuracy} = TP+TN\ /TP+FP+FN+TN$$ [38]

Precision

Precision is the ratio of rightly predicted positive values to the total positive predictions be it false or true. With precision measure, in this study, the question that is tried to be answered in this study is that with total predicted normal logs how many were actually normal? Higher the value of precision, lower is the false positive rate. In this study precision of 0.93 has been achieved with SVM algorithm which is very good precision.

$$\text{Precision} = TP/TP+FP$$ [38]

Recall

Recall or sensitivity of the model is the ratio of rightly predicted positive values to the all value in the real positive class. In this study, the question that is tried to be answered with recall measure is that of all the normal logs, how many was predicted correctly?
For this study a recall of 0.92 is achieved and is very good.

$$\text{Recall} = TP/TP+FN \qquad [38]$$

F1 score

F1 score takes both false positives and false negatives into account and is a weighted average of recall and precision. F1 is usually more useful than accuracy, especially when class distribution is uneven. Sometimes the cost of making wrong prediction can be very different, for e.g. in the case of this study ‘False positive’ kind of predictions can be extremely costly as this will not alert the user on a probable anomaly. In such cases where cost of false positives and false negatives vary a lot, attention must be paid to both Precision and Recall. In this study F1 score of 0.93 is achieved with SVM algorithm.

$$\text{F1 Score} = 2 \times (\text{Recall} \times \text{Precision}) / (\text{Recall} + \text{Precision}) \qquad [38]$$

Multinomial Naive Bayes Algorithm

The Naive Bayes algorithm is a simple probability based algorithm based on Bayes theorem which is widely used in text classification. It has very naive independence assumptions [37] meaning that it assumes that every feature of the dataset contributes independently to the probability of a classification, even though there could be correlations between the features.

Multinomial Naive Bayes is an algorithm with small variation on Naive Bayes which estimates the conditional probability of a single token given a class as the relative occurrence of term t in all the documents belonging to class c as shown below:

$$P(t|c) = \frac{T_{ct}}{\sum_{t' \epsilon v} T_{ct'}} \quad , \qquad [37]$$

where T_{ct} is the total count of term t in all documents of class c.

Below listing list 1 shows the main assumptions done by Multinomial Naive Bayes algorithm.

- Bag of words assumption – assumes the document is just a bunch of words. Position of the word does not matter.
- Conditional independence – assumes that the features are independent of the class.

List1[37] Pseudocode for Multinomial NB

As seen in the pseudocode MNB takes into account the frequency of occurrence of term t in training documents from class c, counting multiple occurrences.

Support Vector Machine Algorithm

SVM is a supervised ML algorithm which is frequently used in classification problems but can also be used for regression problems. In this algorithm, each data item is plotted as a point in n-dimensional feature space, n being the number of features present. Value of each feature is represented by a specific coordinate. These coordinates are called support vectors. Classification is performed by finding the hyper-plane that can segregate the two classes properly [40] as shown in figure 4 below. This hyperplane or border line is called support vector machine.

SVM algorithm is very efficient in dealing with outliers and can find a hyperplane by ignoring some outliers. An outlier is a coordinate that is situated very far from the other coordinates belonging to its class. SVM performs a very interesting kernel trick on the data. Kernels are functions which transform low dimensional input feature space to a higher dimensional space. This kernel trick is very valuable in case of non-linear classification problem. In other words, it performs some complex data transformations to classify the data based on the labels we have defined.

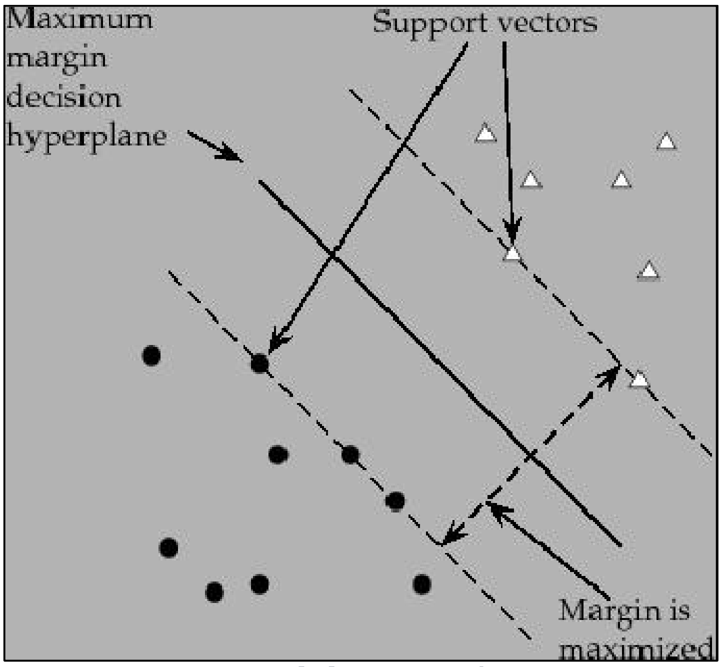


Figure 4 [42] SVM Hyper-Plane

Binary support vector machines are classifiers which classify data points into two categories. Each data point corresponds to an n-dimensional vector and they belong to either of the two classes. A linear classifier splits them with a hyper-plane. In order to achieve maximum separation between the two classes, SVM selects the hyperplane which has the largest margin. Margin is the distance between the optimal hyperplane and the training data point closest to the hyperplane. When this distance is taken on both sides of hyperplane, it forms a region where no data points will be present. Intention of SVM is to make this region as big as possible, and hence optimal hyperplane is selected in such a way that this goal is achieved [41].

Random Forest Algorithm

Random Forest algorithm, also called a random decision forests algorithm, is a popular supervised learning algorithm. This algorithm works well for both classification and regression problems [44].

Random forest algorithm builds multiple random uncorrelated decision trees and unifies them together to get a more stable and correct prediction. The output class is typically the class that has appeared more number of times as the decision result class [43]. Figure 5 shows the pictorial representation of a random forest with 2 decision trees.

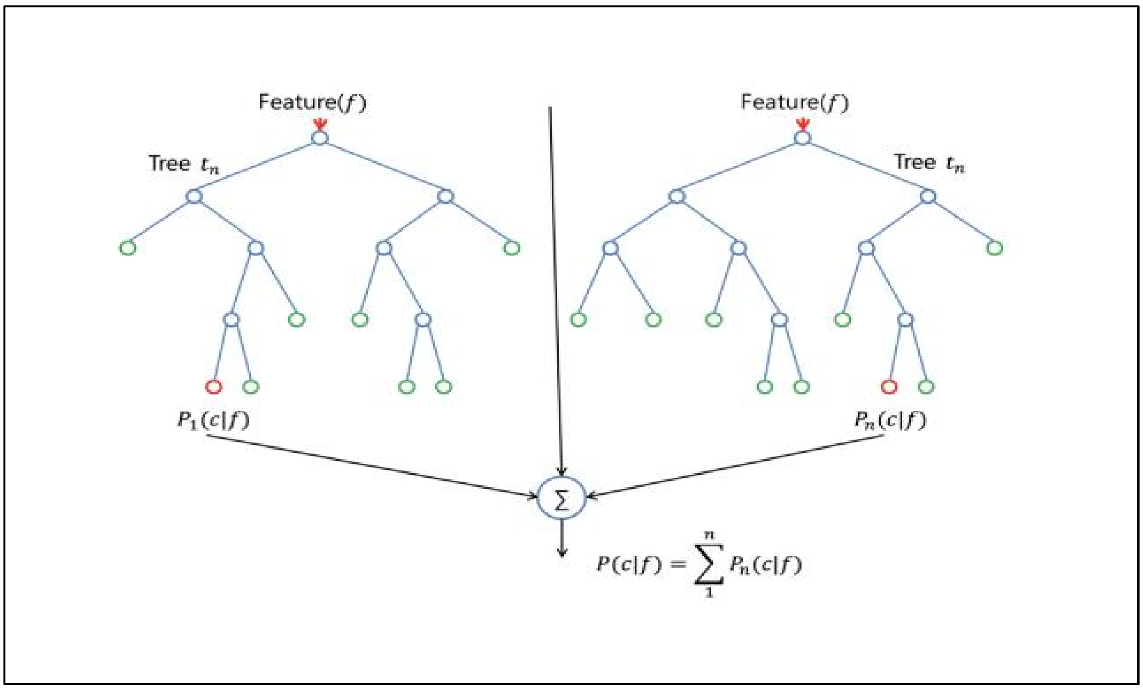


Figure 5 [44] Random forest with 2 decision trees

The random-forest algorithm passes additional randomness into the learning model while growing the trees. As an alternative of finding the best feature while parting a node, it looks for the best features in a random subgroup of features. This process brings in a wide diversity and randomness [16]. Hence in this algorithm, while splitting a node, only random subgroup of features is taken into consideration. Random forests creates multiple decision trees and trains them on multiple parts of the same dataset. These deep decision trees are then averaged with the goal of reducing the variance [43].

K Nearest Neighbors Algorithm

K nearest neighbor's algorithm is a simple supervised learning algorithm that trains on all available cases and stores them and classifies a new case based on distance function between this new case and the stored cases. This algorithm works by classifying a new case on the basis of majority votes of its neighbors. Thus a case is assigned to a class most common among its 'K' nearest neighbors (found using a distance function).

The pseudocode for KNN algorithm is given by list 2:

```
K-Nearest Neighbor
Classify(X,Y,x)  // X: training data ,Y: labels of X, x: new unknown data
for i= 1 to n do
    Compute Euclidean Distance d( $X_i$ , x)
end for
Compute set I containing indices for k smallest distances d($X_i$,x)
Return majority label for { $Y_i$ where i ε I }
```

List 2[45] Pseudocode for KNN Algorithm

The main challenge is using this algorithm for training a model is finding an optimal value of 'K'. Optimal value of 'K' could be found using cross validation of the model. Generally higher values of 'K' are good to reduce the overall noise. If 'K' is too small like 1 there is a risk of overfitting the model [45].

Dataset Description

EC generated server log files has been used as dataset for this study. These server logs were generated based on specific configuration and scenarios. Because of the fact that each log had to be labelled later for training the ML algorithm, any random log file could not be used for this study. Hence log generation was a major task in this thesis. The total dataset of size 638 MB consisting of only EC server log files was generated and collected from January through May. Out of this 412 MB files were used to train the model and rest was used to test the accuracy of the model. More details about EC server log files can be seen in Chapter 4 - methodology.

Chapter 3 – Framework

Overview

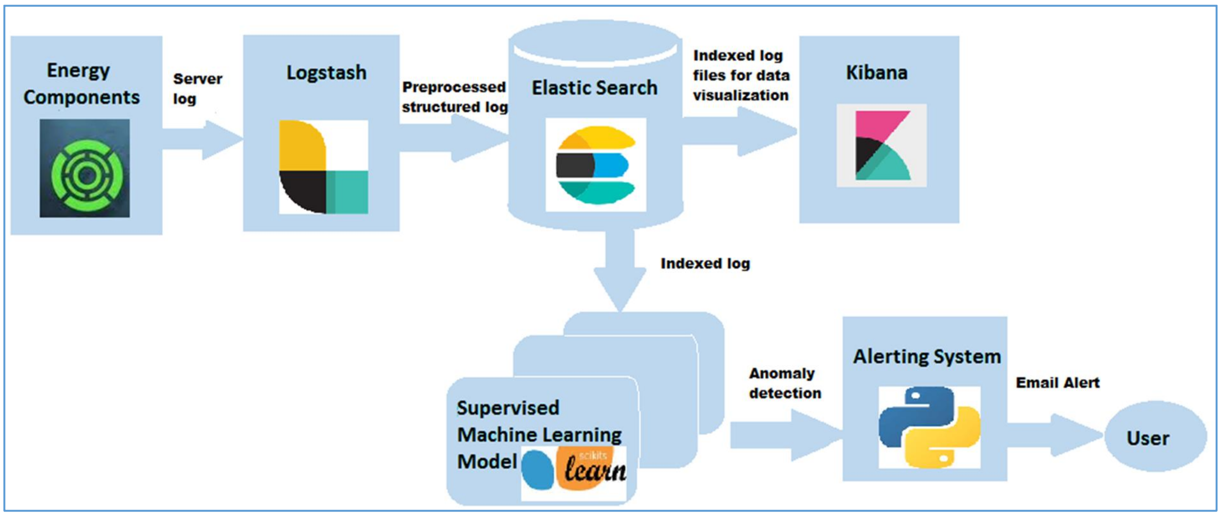


Figure 6 Framework of the prototype

Figure 6 explains the high level framework of the prototype developed for this thesis. The main components in the health check analyzer framework are:

- Energy Components
- Logstash
- Elasticsearch
- Kibana
- Supervised Machine learning model
- Alerting system

Each of the above component and their setup for this prototype is described in detail in the below section.

Energy Components (EC)

All oil and gas operators must retain records of extractions of oil, gas and other byproducts (together called hydrocarbons) from the reservoirs along with maintaining the record of the amount of hydrocarbons spent, re-injected or lost, and the details of transportation and delivery of it to customers and other facilities. This is called hydrocarbon accounting, and Tieto A.S provides this solution in the form of a product called Energy Components [1].

Energy Components product suite gives exceptional support for production operations, gas plants, terminals, LNG facilities, pipeline transportation, cargo shipping, invoice generation, billing support and more. EC strategy is to develop one unified solution from reservoirs to financial accounting for the upstream business, and the EC portfolio has six completely integrated solutions on top of same underlying database [1].

The six modules that EC comprises of are:

- **EC-Production** – This module mainly deals with production management, hydrocarbon allocation and reporting.
- **EC-Transport** – This module covers the support of transporting hydrocarbons either through pipeline systems, transport trucks or cargo vessels.
- **EC Sales –** This module provides the support to sales organization for sales of hydrocarbons by managing gas sales contracts for them.
- **EC Revenue –** This module provides functionality for invoicing and accounting of any data applicable for revenue recognitions
- **EC NOV –** This module provides support for handling Non-Operated production data.
- **EC PSA –** This module covering handling of profit sharing agreement calculations.

EC is based on modern web architecture, referred to as the J2EE architecture using the XML standard and the Oracle RDBMS. Figure 7 outlines the conceptual model for EC Framework.

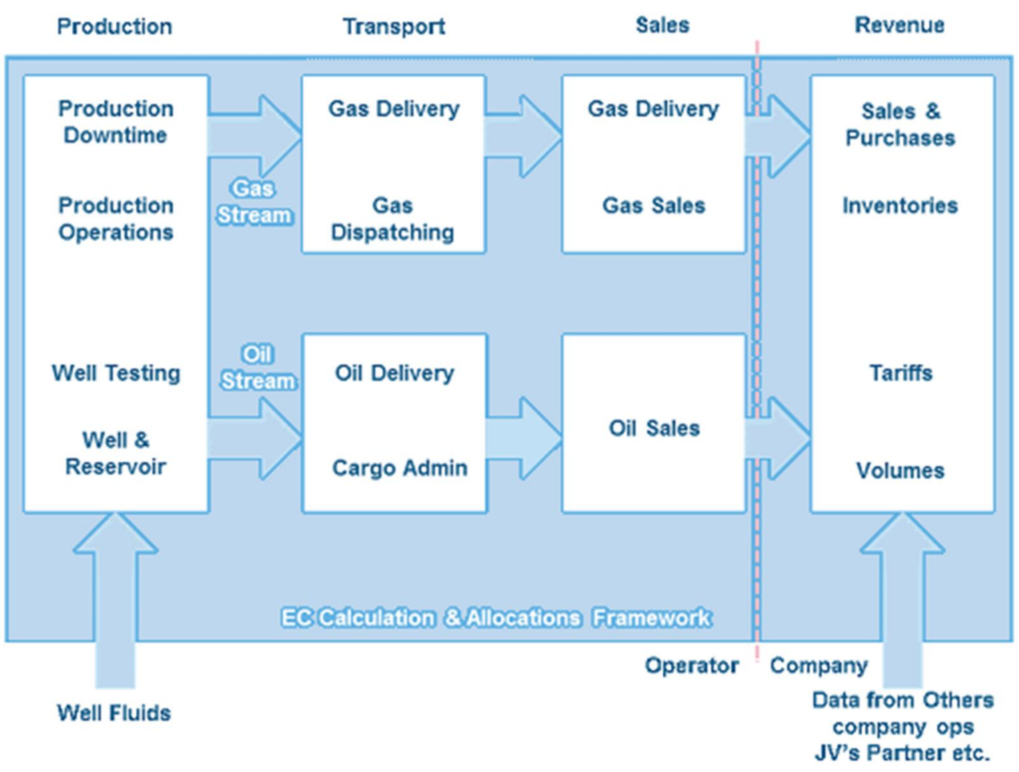


Figure 7[1] Energy Components Product Suite

EC runs on wildfly application server which is written in Java. During the normal operation of the product, it generates log4j server logs to log all the major events happening during the day. These logs withhold a lot of information about the system's overall well-being.

ELK Stack

ELK is a robust and efficient open source solution to search, analyze and visualize log data [19]. It comprises of three open source projects - Elasticsearch, Logstash, and Kibana [17].

Elasticsearch is a NoSQL database which is based on search engine called Lucene used for search and analytics [17]. Logstash is a server-side pipeline which is used for managing logs and events. Logstash ingests data from different sources (in this case from Energy components application), performs various transformations on this logs data, and sends this data to various targets or to a stash or database like Elasticsearch. Kibana acts like a visualization layer on Elasticsearch [19]. This helps users to visualize the data in Elasticsearch using bar graphs or charts. Each of the components of ELK stack are described in the below sections in detail.

The different modules in the ELK Stack are intended to interact and work nicely with each other without the need of much additional configuration. However, the construction and configuration of the stack largely depends on the specific use case of the user. The classic ELK stack architecture looks like the figure 8.

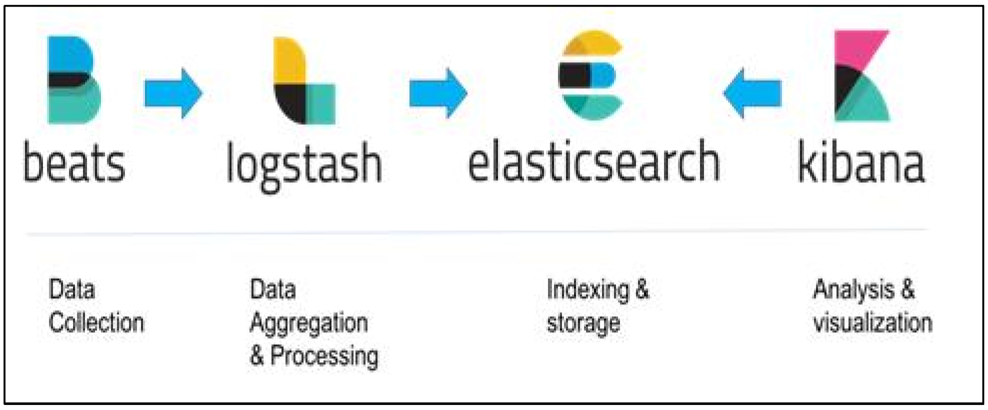
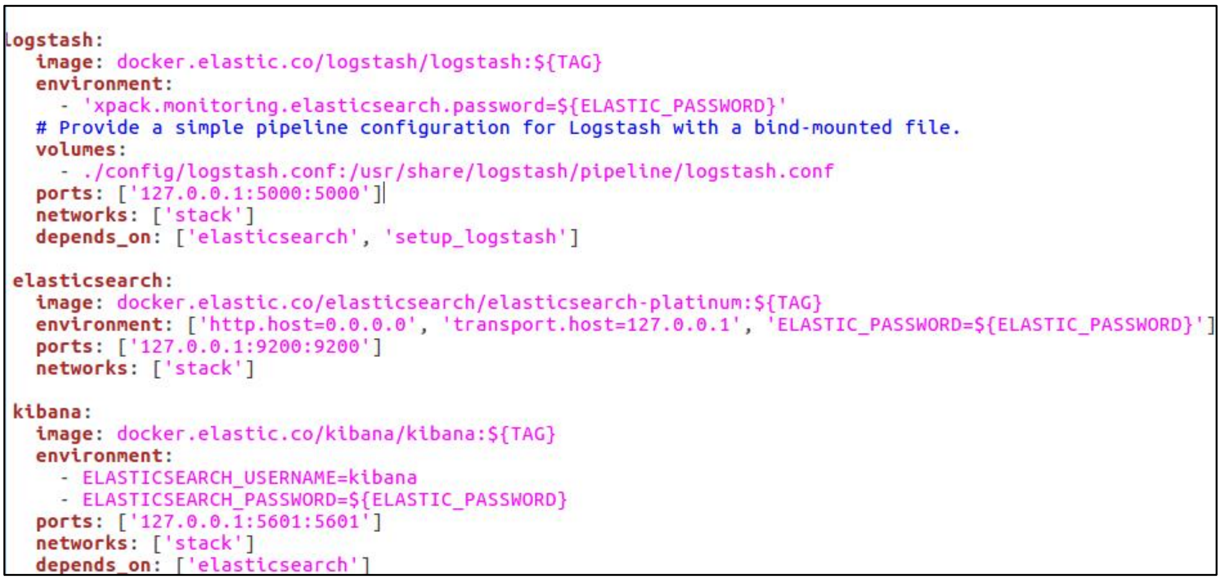


Figure 8 [25] ELK Stack Architecture

ELK Stack Setup

The first step to have an ELK stack up and running is to set up a virtual machine running Linux OS. The benefit with using Linux for running ES is that most of the optimization works for ES are focused in Linux. Also most of the documentation including the official one on Elastic's site assumes user is using Linux to run Elasticsearch.

As the entire ELK stack is implemented in Java and requires a java run time environment for functioning, Java 8 is installed in the virtual machine. Since the ELK stack runs well with Docker and Docker compose, the same were installed and finally the ELK stack. The configuration of Docker-compose was altered so that the output of Logstash is directed to Elasticsearch and Kibana reads from Elasticsearch. The good thing about using docker compose is it fires up the elk stack altogether and hence it is not required to start up Elasticsearch , Kibana and Logstash individually. List 3 shows the configuration of the docker-compose.yml file created for this purpose:

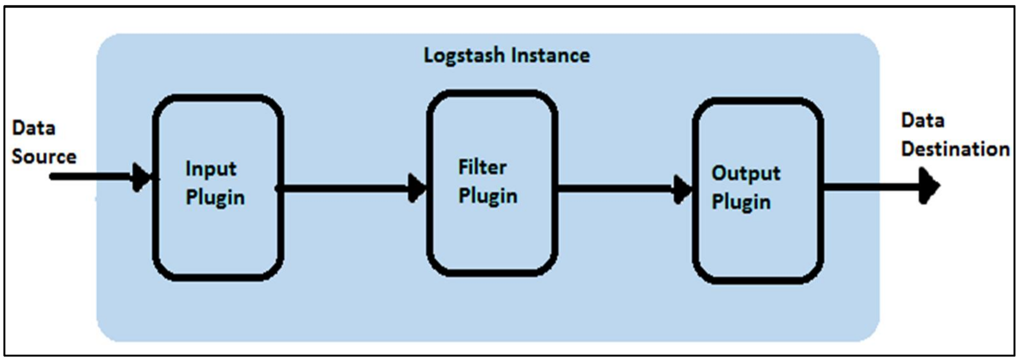


```
logstash:
  image: docker.elastic.co/logstash/logstash:${TAG}
  environment:
    - 'xpack.monitoring.elasticsearch.password=${ELASTIC_PASSWORD}'
  # Provide a simple pipeline configuration for Logstash with a bind-mounted file.
  volumes:
    - ./config/logstash.conf:/usr/share/logstash/pipeline/logstash.conf
  ports: ['127.0.0.1:5000:5000']
  networks: ['stack']
  depends_on: ['elasticsearch', 'setup_logstash']

elasticsearch:
  image: docker.elastic.co/elasticsearch/elasticsearch-platinum:${TAG}
  environment: ['http.host=0.0.0.0', 'transport.host=127.0.0.1', 'ELASTIC_PASSWORD=${ELASTIC_PASSWORD}']
  ports: ['127.0.0.1:9200:9200']
  networks: ['stack']

kibana:
  image: docker.elastic.co/kibana/kibana:${TAG}
  environment:
    - ELASTICSEARCH_USERNAME=kibana
    - ELASTICSEARCH_PASSWORD=${ELASTIC_PASSWORD}
  ports: ['127.0.0.1:5601:5601']
  networks: ['stack']
  depends_on: ['elasticsearch']
```

List 3 Docker Compose Configuration

Logstash

Logstash is a data collection tool that is capable of reading log messages from various sources like messaging queues, HTTP, a port or other logging tools. Since most of the time the inputs from these sources are unstructured or have their specific characteristic structure, Logstash normalizes these input messages and gives them a structure by bringing them in a consistent form. Regardless of the kind of input event, Logstash transforms it into a JSON-like event, containing key-value sets [31]. Thus, any incoming message can be enhanced with additional information, e.g. a debug level or timestamp, and its contents can also be reduced or modified. After preprocessing a message gets directed to one or more destinations. In this case the output of Logstash gets directed to Elasticsearch. Figure 9 descriptively summarizes the classic Logstash workflow.



Figur 9 Logstash Framework

Elasticsearch

Elasticsearch is a full-text search and analytics engine based on Apache Lucene [28]. It is a high performance data base which allows searching and exploring huge volumes of data in real time (almost), something which is difficult to achieve in the conventional SQL database.

In this thesis, ES functions as a storage for logging messages sent to it by Logstash. Just like in conventional databases, ES also provided ways to query for various patterns in the logs (e.g. high exception frequency) thereby facilitating discovery of problems almost in real-time [17]. To achieve fault tolerance and reliability, ES comes with built-in tools which facilitate sharding, clustering and replication.

Kibana

Kibana is a highly configurable open source visualization tool for Elasticsearch data. It allows to import logs from Elasticsearch and has the ability to depict the results in an understandable and detailed way. It offers lots of illustration possibilities like bar charts, tables and diagrams.

Kibana makes it easy to comprehend large volumes of data. Its browser-based interface allows users to swiftly make and share dynamic dashboards which display modifications to Elasticsearch queries in almost real time [27].

Scikit-learn

Scikit-learn is freely available software machine learning library, mostly coded in Python and Cython. It is very useful for modelling medium-scale supervised and unsupervised problems.

.The main advantages of using scikit learn for modelling is its ease of use for beginners. Its performance, ample documentation, and good API consistency are also good reasons for choosing scikit learn for this study [29]. Scikit learn features a lot of classification, clustering and regression algorithm like random forest, gradient boosting, support vector machines, K nearest neighbor etc. and is designed to work in cooperation with Python’s scientific and numerical libraries NumPy and SciPy.

Alerting Mechanism

The smtplib module in Python has an SMTP client session object which can be used to send mail to any machine connected to internet with an SMTP or ESMTP listener daemon [32]. In this thesis if the classification of a new log data is made into ‘Error’ class, then an e- mail alert is generated and sent to user saying please look into the logs as there is something suspicious in the generated logs.

Chapter 4 - Methodology

This chapter explains the different steps used in the methodology of log analysis to achieve the goals of this thesis.

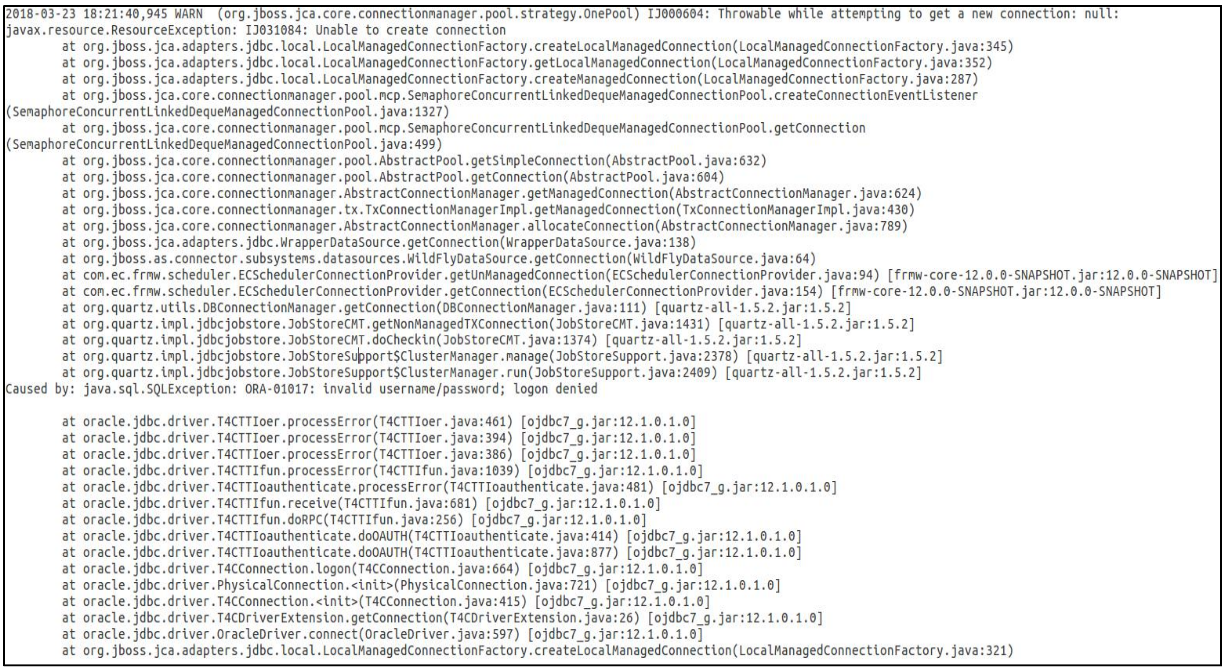
Log Collection

EC systems generates huge server logs to record the state of the system, major events and to log runtime details. Each log line comprises of a timestamp, class information, log level and a log message indicating what event had happened. This information is very valuable and hence used for log analysis purpose. Therefore, for this thesis, the very first step was to generate many scenario based EC server log files.

A dedicated EC application running on an internal Tieto server was used to configure specific scenarios and generate normal and anomalous logs for training purpose. Detailed study of many modules inside EC was done to generate such scenario based logs. Many EC module experts were contacted to gather specific information for the same. This information from subject matter experts was needed because it will help later to label these log files for training purpose. Figure 10 below shows typical EC server Log message for one timestamp.

As it can be seen, one log message (for one timestamp) has considerable amount of information in each log line like:

- Date,
- Timestamp,
- Log level,
- Class name
- Message text.
- Stack trace
- port



```
2018-03-23 18:21:40,945 WARN  (org.jboss.jca.core.connectionmanager.pool.strategy.OnePool) IJ000604: Throwable while attempting to get a new connection: null:
javax.resource.ResourceException: IJ031084: Unable to create connection
	at org.jboss.jca.adapters.jdbc.local.LocalManagedConnectionFactory.createLocalManagedConnection(LocalManagedConnectionFactory.java:345)
	at org.jboss.jca.adapters.jdbc.local.LocalManagedConnectionFactory.getLocalManagedConnection(LocalManagedConnectionFactory.java:352)
	at org.jboss.jca.adapters.jdbc.local.LocalManagedConnectionFactory.createManagedConnection(LocalManagedConnectionFactory.java:287)
	at org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreConcurrentLinkedDequeManagedConnectionPool.createConnectionEventListener
(SemaphoreConcurrentLinkedDequeManagedConnectionPool.java:1327)
	at org.jboss.jca.core.connectionmanager.pool.mcp.SemaphoreConcurrentLinkedDequeManagedConnectionPool.getConnection
(SemaphoreConcurrentLinkedDequeManagedConnectionPool.java:499)
	at org.jboss.jca.core.connectionmanager.pool.AbstractPool.getSimpleConnection(AbstractPool.java:632)
	at org.jboss.jca.core.connectionmanager.pool.AbstractPool.getConnection(AbstractPool.java:604)
	at org.jboss.jca.core.connectionmanager.AbstractConnectionManager.getManagedConnection(AbstractConnectionManager.java:624)
	at org.jboss.jca.core.connectionmanager.tx.TxConnectionManagerImpl.getManagedConnection(TxConnectionManagerImpl.java:430)
	at org.jboss.jca.core.connectionmanager.AbstractConnectionManager.allocateConnection(AbstractConnectionManager.java:789)
	at org.jboss.jca.adapters.jdbc.WrapperDataSource.getConnection(WrapperDataSource.java:138)
	at org.jboss.as.connector.subsystems.datasources.WildFlyDataSource.getConnection(WildFlyDataSource.java:64)
	at com.ec.frmw.scheduler.ECSchedulerConnectionProvider.getUnManagedConnection(ECSchedulerConnectionProvider.java:94) [frmw-core-12.0.0-SNAPSHOT.jar:12.0.0-SNAPSHOT]
	at com.ec.frmw.scheduler.ECSchedulerConnectionProvider.getConnection(ECSchedulerConnectionProvider.java:154) [frmw-core-12.0.0-SNAPSHOT.jar:12.0.0-SNAPSHOT]
	at org.quartz.utils.DBConnectionManager.getConnection(DBConnectionManager.java:111) [quartz-all-1.5.2.jar:1.5.2]
	at org.quartz.impl.jdbcjobstore.JobStoreCMT.getNonManagedTXConnection(JobStoreCMT.java:1431) [quartz-all-1.5.2.jar:1.5.2]
	at org.quartz.impl.jdbcjobstore.JobStoreCMT.doCheckin(JobStoreCMT.java:1374) [quartz-all-1.5.2.jar:1.5.2]
	at org.quartz.impl.jdbcjobstore.JobStoreSupport$ClusterManager.manage(JobStoreSupport.java:2378) [quartz-all-1.5.2.jar:1.5.2]
	at org.quartz.impl.jdbcjobstore.JobStoreSupport$ClusterManager.run(JobStoreSupport.java:2409) [quartz-all-1.5.2.jar:1.5.2]
Caused by: java.sql.SQLException: ORA-01017: invalid username/password; logon denied

	at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java:461) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java:394) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIoer.processError(T4CTTIoer.java:386) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIfun.processError(T4CTTIfun.java:1039) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIoauthenticate.processError(T4CTTIoauthenticate.java:481) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIfun.receive(T4CTTIfun.java:681) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIfun.doRPC(T4CTTIfun.java:256) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIoauthenticate.doOAUTH(T4CTTIoauthenticate.java:414) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CTTIoauthenticate.doOAUTH(T4CTTIoauthenticate.java:877) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CConnection.logon(T4CConnection.java:664) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.PhysicalConnection.<init>(PhysicalConnection.java:721) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CConnection.<init>(T4CConnection.java:415) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.T4CDriverExtension.getConnection(T4CDriverExtension.java:26) [ojdbc7_g.jar:12.1.0.1.0]
	at oracle.jdbc.driver.OracleDriver.connect(OracleDriver.java:597) [ojdbc7_g.jar:12.1.0.1.0]
	at org.jboss.jca.adapters.jdbc.local.LocalManagedConnectionFactory.createLocalManagedConnection(LocalManagedConnectionFactory.java:321)
```

Figure 10 EC Server Log

The server logs generated by Energy components is directed to a port ‘5000’ using the netcat command as shown in list 4:

```
cat server.log | nc localhost 5000
```

List 4. Netcat Command

Log Parsing and Preprocessing

Logstash is configured to listen to the port 5000 and ingest incoming data. The configuration of the logstash input plugin is specified in list 5.

```
input {
       tcp {
                port => 5000
           # fix up multiple lines in log output into one entry
                codec => multiline {
                  pattern => "^20[0-9]{2}-"
                negate => true
                what => "previous"
                  }
             }
       }
```

List 5 Logstash Input Plugin Configuration

Also as the above configuration code depicts, the incoming data is modified in such a way that multiple lines for the same timestamp is combined to one line. That means if Logstash reads the log message, it will first combine all that into one big line, since all of it belongs to one timestamp - 2018-01-18 18:56:56.

But since the incoming message has lot of unwanted information like port number, and also has lot of unstructured useful information, it is needed that more data processing has to be performed. Therefore grok filter is applied to the read log message and split each log line into multiple features like class name, log level, message, timestamp etc. The grok filter gives more structure to the log message. The grok filter configuration for this thesis is shown in the below code snippet. The filter section in this code snippet is for parsing the messages (shown in list 6).

```
## Logstash filter configuration

filter {
   mutate {
     strip => "message"
    }
   grok {
        match => {
"message" => "%{TIMESTAMP_ISO8601:logdate} %{LOGLEVEL:loglevel}    %{NOTSPACE:className}
%{GREEDYDATA:message}"
         }
    }
   date {
     match => ["logdate", "yyyy-MM-dd HH:mm:ss,SSS", "ISO8601"]
     target => "@timestamp"
   }

}
```

List 6 Logstash Filter Plugin Configuration

Now that the log message is more structured, some more cleanup of the log file itself needs to be done.

EC server log files have multiple event loggings of log level ‘info’ and ‘debug’. These event is mostly useful for developers while debugging the code. Hence for this study which is to analyze logs to find anomalous logs, these events would be removed in Logstash. This process considerably reduces the size of the log files and makes it easier to handle them further in the process of analyzing the logs. The code snippet for the same is specified in the below code snippet list 7.

```
# filtered out info and debug statements
output {
 if   "ERROR" in [loglevel] or "SEVERE" in [loglevel] or "WARN"in [loglevel]
{
  elasticsearch {
    hosts    => [ 'elasticsearch' ]
    user     => 'elastic'
    password => 'changeme'
                  }
  }
  }
```

List 7 Logstash Output Plugin Configuration

The log data is then pushed to Elasticsearch which acts like a database to store the data. Elasticsearch adds indices to the read log data which make it easily searchable irrespective of the amount of data to be searched.

Data Visualization and Labeling

Visualization

Once the data is available in Elasticsearch, it can be viewed through Kibana. EC generated logs are preprocessed and filtered using the Logstash grok filter and output plugin as mentioned in previous section. Transformed data can be seen by querying Elasticsearch with simple mlt queries. Another way to check the data is by creating various kibana plots and analyze the data. Figure 11 is one such Kibana visualization chart which was generated and analyzed to see which log levels are present in the content of Elasticsearch.

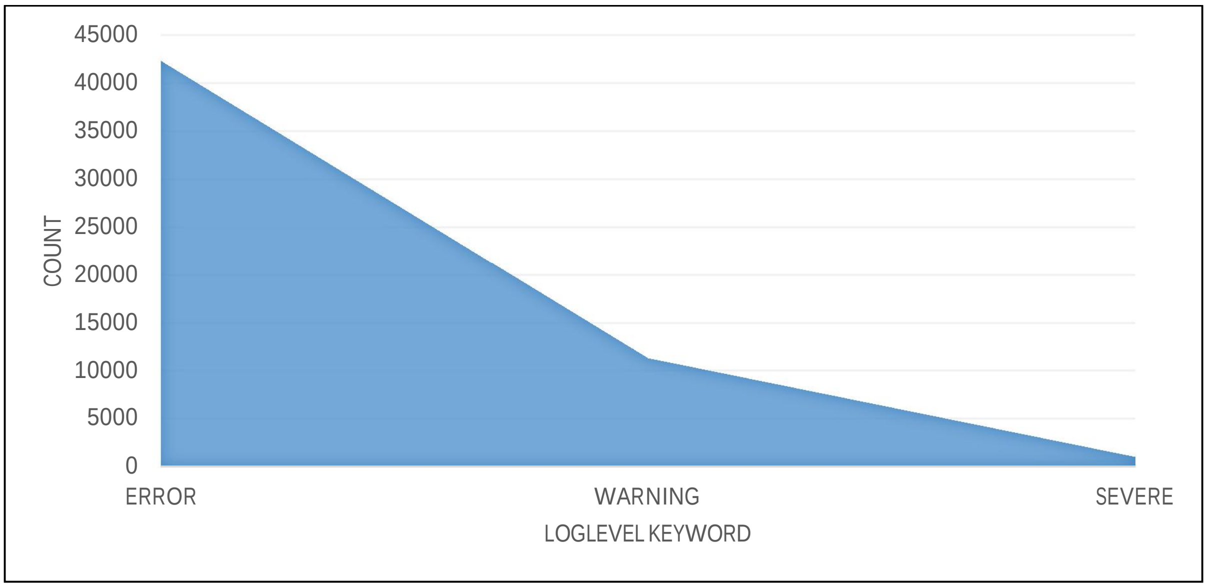


Figure 11 Loglevel Keywords Count

Figure 11 shows that there are no ‘info’ and ‘debug’ type of log levels in the Elasticsearch data. Few more visualizations are attached below to depict the usage of Kibana and to check that all the preprocessing really worked. Figure 12 is a kibana visualization of log data in April.

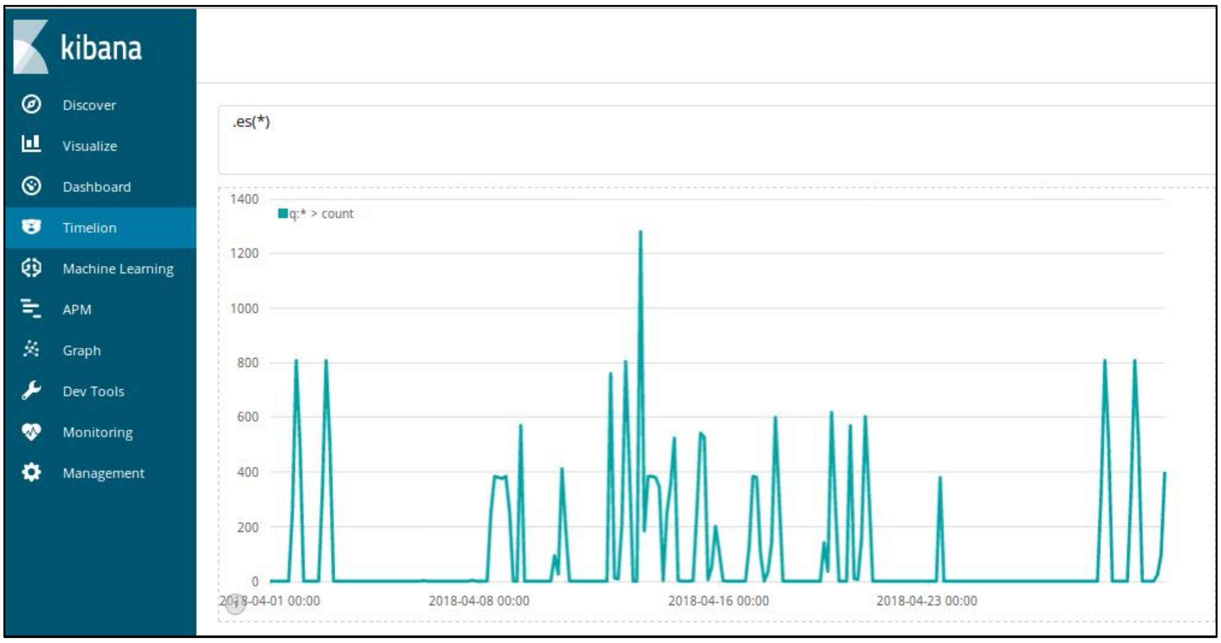
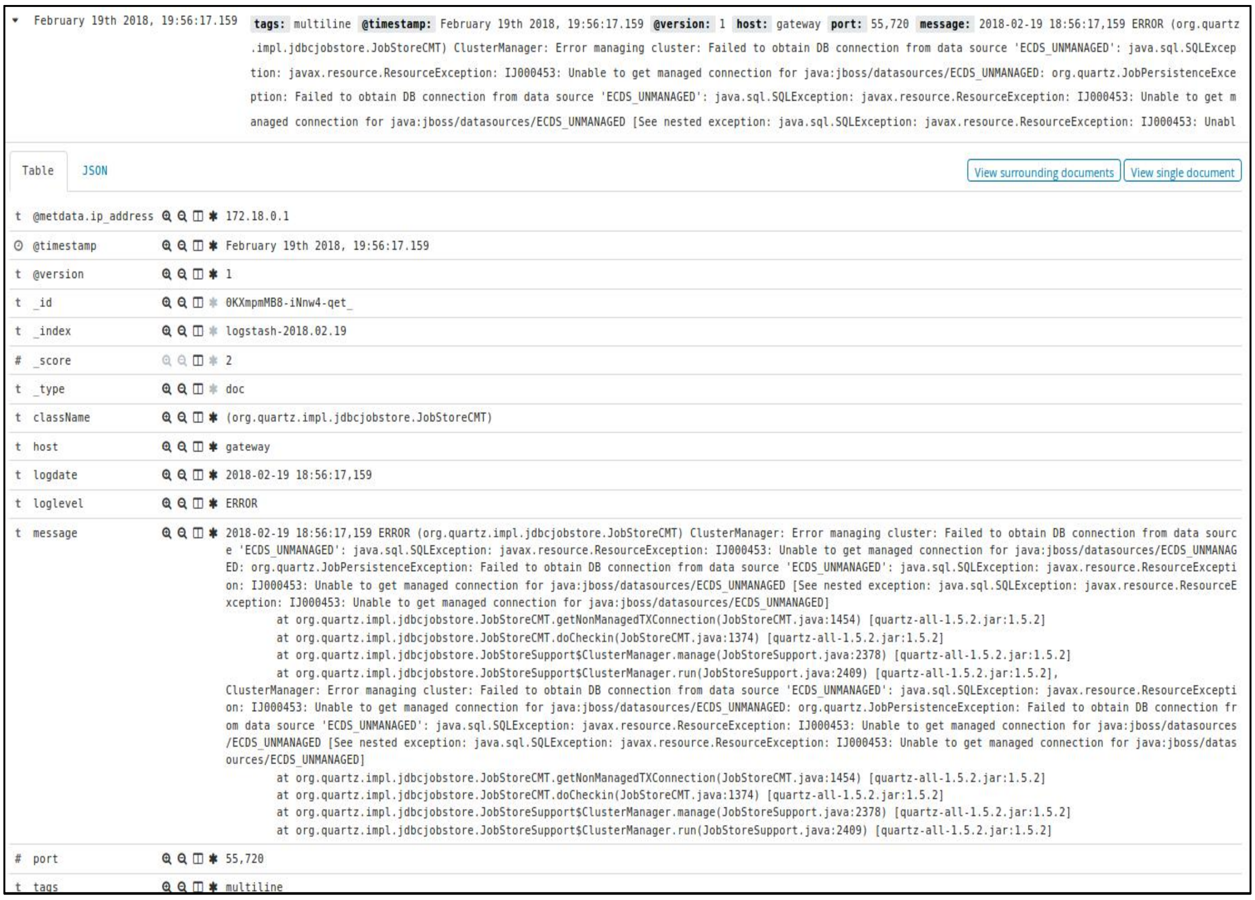


Figure 12. Kibana Visualization: April data from Elasticsearch

Further Figure 13 shows how each line is split to different tags, which is done by the grok filter of Logstash. One log message has ip_address, timestamp , version , id , index , score , type, class name , host , log date , log level , message , port and tags. These tags are very useful for performing different visualizations in Kibana as they hold a lot of information within them.

Interesting thing to note here is that all this split of a log message information to different tags was done by Logstash filter plugin configuration. If this was supposed to be done by explicit programming using NLP (natural language processing tools), it would have been a lot more complex and would have taken a lot of time and effort. This in a way justifies the use of Logstash for this study.



```
▾ February 19th 2018, 19:56:17.159   tags: multiline @timestamp: February 19th 2018, 19:56:17.159 @version: 1 host: gateway port: 55,720 message: 2018-02-19 18:56:17,159 ERROR (org.quartz
                                     .impl.jdbcjobstore.JobStoreCMT) ClusterManager: Error managing cluster: Failed to obtain DB connection from data source 'ECDS_UNMANAGED': java.sql.SQLExcep
                                     tion: javax.resource.ResourceException: IJ000453: Unable to get managed connection for java:jboss/datasources/ECDS_UNMANAGED: org.quartz.JobPersistenceExce
                                     ption: Failed to obtain DB connection from data source 'ECDS_UNMANAGED': java.sql.SQLException: javax.resource.ResourceException: IJ000453: Unable to get m
                                     anaged connection for java:jboss/datasources/ECDS_UNMANAGED [See nested exception: java.sql.SQLException: javax.resource.ResourceException: IJ000453: Unabl

Table   JSON                                                                                                   View surrounding documents   View single document

t @metdata.ip_address  172.18.0.1
⊙ @timestamp           February 19th 2018, 19:56:17.159
t @version             1
t _id                  0KXmpmMB8-iNnw4-qet_
t _index               logstash-2018.02.19
# _score               2
t _type                doc
t className            (org.quartz.impl.jdbcjobstore.JobStoreCMT)
t host                 gateway
t logdate              2018-02-19 18:56:17,159
t loglevel             ERROR
t message              2018-02-19 18:56:17,159 ERROR (org.quartz.impl.jdbcjobstore.JobStoreCMT) ClusterManager: Error managing cluster: Failed to obtain DB connection from data sourc
                       e 'ECDS_UNMANAGED': java.sql.SQLException: javax.resource.ResourceException: IJ000453: Unable to get managed connection for java:jboss/datasources/ECDS_UNMANAG
                       ED: org.quartz.JobPersistenceException: Failed to obtain DB connection from data source 'ECDS_UNMANAGED': java.sql.SQLException: javax.resource.ResourceExcepti
                       on: IJ000453: Unable to get managed connection for java:jboss/datasources/ECDS_UNMANAGED [See nested exception: java.sql.SQLException: javax.resource.ResourceE
                       xception: IJ000453: Unable to get managed connection for java:jboss/datasources/ECDS_UNMANAGED]
                               at org.quartz.impl.jdbcjobstore.JobStoreCMT.getNonManagedTXConnection(JobStoreCMT.java:1454) [quartz-all-1.5.2.jar:1.5.2]
                               at org.quartz.impl.jdbcjobstore.JobStoreCMT.doCheckin(JobStoreCMT.java:1374) [quartz-all-1.5.2.jar:1.5.2]
                               at org.quartz.impl.jdbcjobstore.JobStoreSupport$ClusterManager.manage(JobStoreSupport.java:2378) [quartz-all-1.5.2.jar:1.5.2]
                               at org.quartz.impl.jdbcjobstore.JobStoreSupport$ClusterManager.run(JobStoreSupport.java:2409) [quartz-all-1.5.2.jar:1.5.2],
                       ClusterManager: Error managing cluster: Failed to obtain DB connection from data source 'ECDS_UNMANAGED': java.sql.SQLException: javax.resource.ResourceExcepti
                       on: IJ000453: Unable to get managed connection for java:jboss/datasources/ECDS_UNMANAGED: org.quartz.JobPersistenceException: Failed to obtain DB connection fr
                       om data source 'ECDS_UNMANAGED': java.sql.SQLException: javax.resource.ResourceException: IJ000453: Unable to get managed connection for java:jboss/datasources
                       /ECDS_UNMANAGED [See nested exception: java.sql.SQLException: javax.resource.ResourceException: IJ000453: Unable to get managed connection for java:jboss/datas
                       ources/ECDS_UNMANAGED]
                               at org.quartz.impl.jdbcjobstore.JobStoreCMT.getNonManagedTXConnection(JobStoreCMT.java:1454) [quartz-all-1.5.2.jar:1.5.2]
                               at org.quartz.impl.jdbcjobstore.JobStoreCMT.doCheckin(JobStoreCMT.java:1374) [quartz-all-1.5.2.jar:1.5.2]
                               at org.quartz.impl.jdbcjobstore.JobStoreSupport$ClusterManager.manage(JobStoreSupport.java:2378) [quartz-all-1.5.2.jar:1.5.2]
                               at org.quartz.impl.jdbcjobstore.JobStoreSupport$ClusterManager.run(JobStoreSupport.java:2409) [quartz-all-1.5.2.jar:1.5.2]
# port                 55,720
t tags                 multiline
```

Figure 13 Single Log Message Split into Multiple Tags

Labeling

Supervised learning requires good quality labeled training data to guarantee that models can predict or classify with high precision and accuracy. Hence labelling is a mandatory step in this thesis as well.

The first step in this process is to make a python program connect to Elasticsearch and read the indices from there and store them as text files in a folder. A very nice feature of Elasticsearch is its in-built RESTful API which offers a very clean and easy interface for accessing and performing operations on Elasticsearch data.To connect python to Elasticsearch this official low-level Elasticsearch client has been used. This client offers direct mapping from Python to Elasticsearch REST end points [39]. Below code snippet in list 8 show the connection to Elasticsearch from python:

```
from elasticsearch import Elasticsearch
es = Elasticsearch([{'host': 'localhost','port': 9200, 'http_auth':'elastic:changeme'}])
```

List 8 Elastic Search to Python Connection

After connection to Elasticsearch, the log indices in Elasticsearch were read to text file (one index to one text file). Below code snippet in list 9 shows how index named as ‘logstash-2018.05.28’ was written to '2018-05-28.txt' file.

```
indices =es.indices.get_alias().keys()
sorted
myquery={"query": {"match_all": {}}}
res = es.search(index="logstash-2018.05.28", body={"query": {"match_all": {}}})
mylist=res['hits']['hits']
file = open('2018-05-28.txt','w')
for i in range(len(res['hits']['hits'])):
    file.writelines( res['hits']['hits'][i]['_source']['message'])
file.close
```

List 9 Reading an Index from ES

After reading all the log indices as text files, the process of manually sorting them into two categories was performed. This process is called labeling. In this study logs have been classified into two categories: Normal and Erroneous. Since the logs were generated based on scenarios, it was known whether they normal or erroneous from the beginning and this information was helpful in labeling them. The classified logs are put into two folders Normal and Erroneous. The idea is that all the files in the folder named ‘Normal’ get the label normal and the ones in folder names ‘Erroneous’ get the label ‘Erroneous’.

Talking about labeling, this question arises:

a) If there are only warning, error type of log levels in the text file, will it not always be categorized as Erroneous? Can a log file having error message be considered normal?

Answer to this is as follows:

- No, if there are only warning and error type of log levels in the text file, it need not always be categorized as Erroneous. There are different types of errors that EC dumps into its server logs. Sometimes these errors are due to a serious anomaly occurring behind the scene like database crashing or Wildfly server going down etc. But sometimes it might be because of simple reason that a piece of code was looking for a table or view which it could not find. The first kind of errors are considered anomaly and hence log files that have these errors would go to Erroneous category folder. While the second type of error is a sql error (generally with low criticality). This is an error occurring due to some scripts missing and user need not be alerted for this. Hence this is categorized into Normal category folder. Hence the category completely depends on the type of error message.

After labeling, about 30 % of the files (15 % from each category) was moved to another folder. This would form the test dataset for this study. These files would not be introduced to the model during training phase and would be used in testing phase to check the accuracy of the model. This is called train/test split.

Feature extraction

The next step in methodology is feature extraction on the data stored in the two folders named Normal and Erroneous respectively. To load the data, load files function in scikit-learn is used. This function can read text files and label them to categories same as subfolder names on which the files are stored. The code for this is shown in the code snippet in list 10.

```
from sklearn.datasets import load_files
log_train=load_files('/home/anju/Desktop/Project2018/serverlogstotrain',description=None ,
categories =None,load_content=True,shuffle=True, encoding='windows-1252',decode_error
='strict',random_state=15)
```

List 10 Load files to train

After loading the training files into log train variable, they need to be further converted into numerical feature vectors, whereby machine learning models can be applied. This process of converting a collection of words in a text file to numerical feature vectors is called a vectorization [35].

The strategy used in this study is called Bag of Words representation. Here the text files are described by word occurrences where the position of the word in the document has no relevance. Then, for each text file (log), a feature vector is generated by tokenizing the words, counting the number of times each word occurs in the text and normalizing the count. This feature vector denotes the occurrence frequency of each token. All feature vectors together can form a feature matrix [35]. List 11 shows the code for the same.

```
# Features extraction from text files
from sklearn.feature_extraction.text import CountVectorizer
count_vectorizer = CountVectorizer()
X_log_train_counts = count_vectorizer.fit_transform(log_train.data)
```

List 11 Count Vectorizer

After this Tf-Idf is found for X_log_train_counts. The methodology of Tf-Idf is elaborated in theoretical background section of this report. Tf-Idf code is shown by list 12.

```
# Finding TF-IDF
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_log_train_tfidf = tfidf_transformer.fit_transform(X_log_train_counts)
```

List 12 Tf-Idf

Training the model

The procedure of training an Supervised machine learning model is performed by introducing a learning algorithm (Machine learning algorithm) to a set of training data to learn from. Then the algorithm tries to generate a mapping function based on the input training data.

The below code in list 13 shows the code of a machine learning model based on multinomial Naive Bayes algorithm.

```
from sklearn.pipeline import Pipeline
text_clf = Pipeline([('vect', CountVectorizer()),('tfidf', TfidfTransformer()),('clf', MultinomialNB(fit_prior=False)),])
text_clf.fit(log_train.data, log_train.target)
```

List 13 Training the model

Testing the model and Alerting

Once training the model has been performed, the test dataset (which had been split earlier) is introduced and tested if the model can classify completely new log files correctly. This process is called testing the learning model. List 14 shows the code for prediction with test dataset using Multinomial Naive Bayes classifier.

```
# Code for testing Multinomial Naive Bayes classifier on test dataset.

import numpy as np
log_test = load_files('/home/anju/Desktop/Project2018/serverlogstotest',description=None , categories =None,load_content=True,shuffle=True, encoding='windows-1252',decode_error ='strict',random_state=42)
docs_test = log_test.data
predicted_NB = text_clf.predict(docs_test)
```

List 14 MNB model

Note that in the above code, same classifier ‘text_clf’ in code list 13, is used for training is used to make prediction for the test dataset.

If the classification of a new log file is made into ‘Erroneous’ class, then an e - mail alert using pythons built in smtp library is generated sent to user saying please look into the logs as there could be an anomaly. List 15 shows the code for email alert.

```
import smtplib

server = smtplib.SMTP('smtp.gmail.com', 587)
server.starttls()
server.login("user2020@gmail.com", "*******")
msg = "Attention!!  Please check the system as we found some Serious errors ”
server.sendmail("user2020@gmail.com", " user2018@gmail.com", msg)
server.quit()
```

List 15 Email Alert

Chapter 5 – Results and Discussion

This chapter covers main experiments done, their results and discussion on the results. All the machine learning models are built in scikit learn using Python programming language and scikit learn's collection of Machine Learning libraries.

The following 4 learning algorithms are used for training and tested for prediction accuracy:

- Multinomial Naive Bayes Algorithm
- Support Vector Machine
- K Nearest Neighbor Algorithm
- Random Forest Algorithm

For each of the above algorithms following performance measures are generated/calculated:

- Prediction accuracy when using count vectorizer
- Prediction accuracy when using stop words
- Prediction accuracy when using Snowball Stemming.
- Prediction accuracy using hashing vectorizer
- Cross validation and plot of learning curve
- Performance matrix with accuracy , precision and F1 factor
- K-fold cross validation with mean and standard deviation plot
- Time to train and test the model

Finally a comparative study of all of them is summarized and best performing algorithm is stated.

Case Study

Case 1: Model based on Multinomial Naive Bayes Algorithm

a. Using count vectorizer and Tf-Idf

When Multinomial Naive Bayes algorithm is used for prediction on the test dataset having 114 files using count vectorizer and Tf-Idf the accuracy achieved is 0.8859649122807017. The count vectorizer managed to extract 9485 features. Table 1 shows the performance matrix obtained for this model using MNB, count vectorizer and Tf-Idf.

	precision	recall	f1-score	support
Erroneous	0.95	0.62	0.75	32
Normal	0.87	0.99	0.93	82
Avg/Total	0.89	0.89	0.88	114

Table 1 Performance Matrix – MNB – using Count Vectorizer and Tf-Idf

For Multinomial Naive Bayes algorithm 89 % is a good accuracy to start with.

b. Using count vectorizer with stop word removal and Tf-Idf

In this case the effect of count vectorizer with stopwords removed along with Tf-Idf is studied. It is found that stop words removal has helped the model and increased the accuracy to 0.916140350877193. Also the time to train the model was much less than before.

c. Using stemming vectorizer and Tf-Idf

Next study was conducted using another vectorizer called Stemming Count Vectorizer, which is nothing but a vectorizer that performs stemming on all the tokens (see theoretical background–Chapter 2). It is seen that the stemming does not increase accuracy for this model and the accuracy continuous to be the 0.916140350877193.

d. Using hashing vectorizer and Tf-Idf

Finally the model is tested with hashing vectorizer. This vectorizer managed to extract 1048576 features from the given 114 files. The Accuracy of Naive Bayes prediction using Hashing vectorizer increased to 0.9298245614035088 although it is observed that longer time is taken to train and to test. The performance matrix using Hashing vectorizer is given by table 2.

	precision	recall	f1-score	support
Erroneous	0.96	0.78	0.86	32
Normal	0.92	0.99	0.95	82
Avg/Total	0.93	0.93	0.93	114

Table 2 Performance matrix -MNB- using Hashing Vectorizer and Tf-Idf

The line plot in figure 14 depicts how the accuracy of the model changed using different vectorizers.

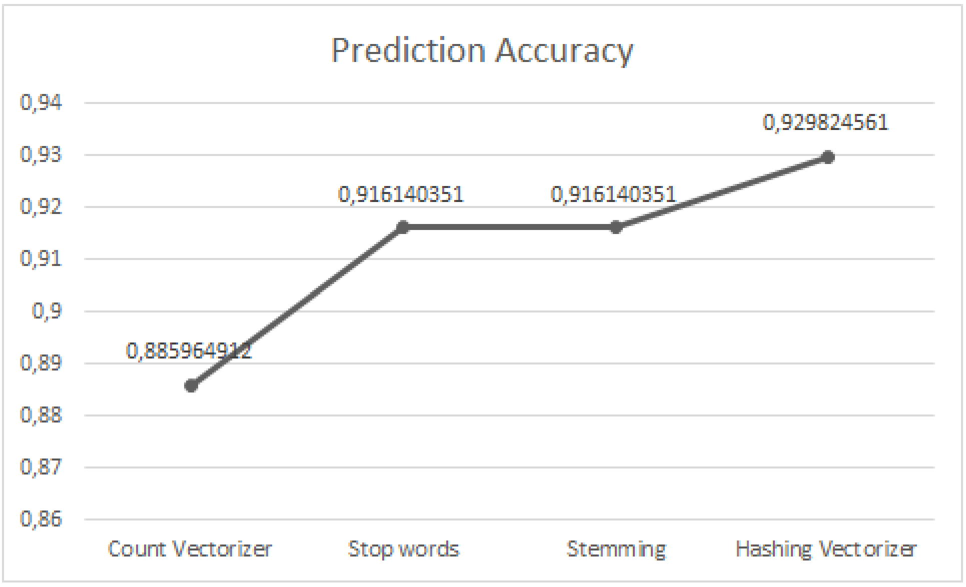


Figure 14 Prediction Accuracy MNB

According to the plot in figure 14, hashing vectorizer gives the best accuracy score for the model based on Multinomial Naive Bayes Algorithm.

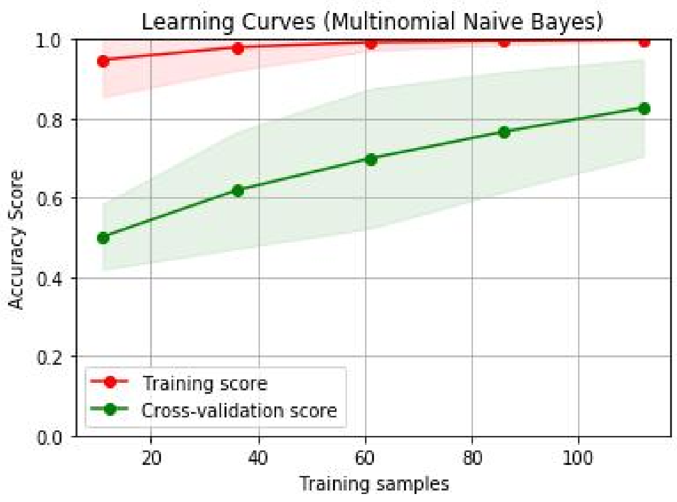


Figure 15. Learning curve MNB using count vectorizer and Tf-Idf

Learning curve of model based on Multinomial Naive Bayes and count vectorizer is plotted in figure 15. The figure shows that training score was a bit lower than 1 in the beginning but after some time it came close to maximum and then it saturates. However, the cross-validation (cv) score depended on the training samples - as the number of training sample increased, so did the cv score. The gap between the two curves imply that more training samples need to be provided to fill in those gaps so that the two curves coincide.

Case 2: Model based on Support Vector Machine Algorithm

a. Using count vectorizer and Tf-Idf

When Support Vector Machine algorithm is used on the test dataset having the same 114 files (as in previous case study) using count vectorizer and Tf-Idf the accuracy achieved is 0.9210526315789473. This is higher than MNB model with count vectorizer. Table 3 shows the performance matrix obtained for this model using SVM, count vectorizer and Tf-Idf.

	precision	recall	f1-score	support
Erroneous	0.81	0.94	0.87	32
Normal	0.97	0.91	0.94	82
Avg/Total	0.93	0.92	0.92	114

Table 3 Performance Matrix – SVM– using Count Vectorizer and Tf-Idf

b. Using count vectorizer with stop word removal and Tf-Idf

This case is conducted to study the effect of stop words removal along with count vectorizer and Tf-Idf. It is found that stop words removal has helped the model a bit and increased the accuracy to 0.9385964912280702. This is the best score for accuracy achieved so far.

c. Using stemming vectorizer and Tf-Idf

This study uses Stemming Count Vectorizer on this model. Interestingly, stemming does not increase accuracy but on the other hand decreases the accuracy for this model to 0.9298245614035088. This could be because different words with same root had helped the algorithm better in classification and now after using snowball stemmer those words have been merged. Accuracy is still better than using simple count vectorizer on this model (case ‘a’), but not as good as count vectorizer with stop word removal and Tf-Idf (case ‘b’).

d. Using hashing vectorizer and Tf-Idf

Finally the model is tested with hashing vectorizer. The Accuracy of model using SVM and Hashing vectorizer is 0.9375464993681742 which is very close to case 'b' for SVM. The performance matrix using Hashing vectorizer is given by table 4.

	precision	recall	f1-score	support
Erroneous	0.86	0.94	0.90	32
Normal	0.97	0.94	0.96	82
Avg/Total	0.94	0.94	0.94	114

Table 4 Performance matrix - SVM-using Hashing Vectorizer and Tf-Idf

The line plot in figure 16 depicts how the prediction accuracy of the model changed using different vectorizers.

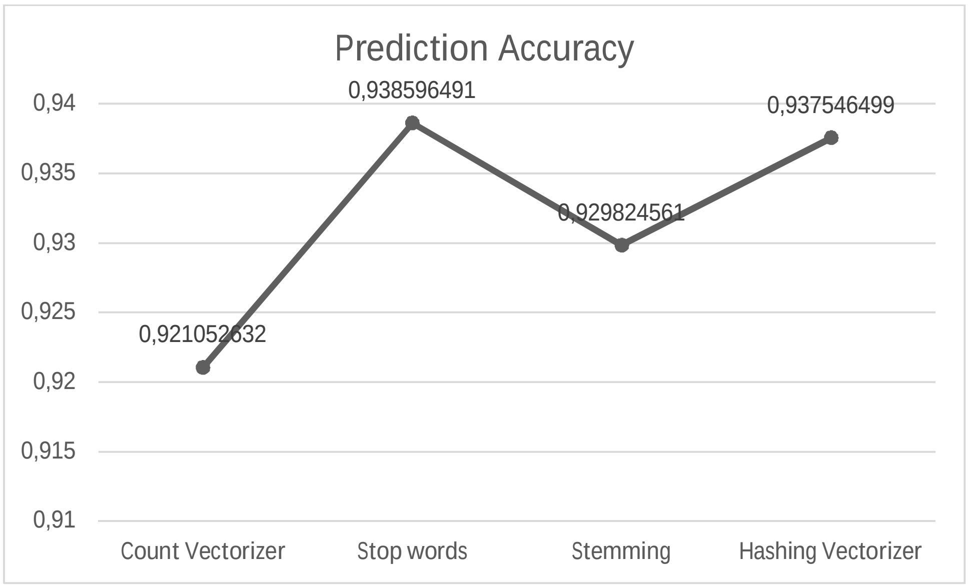


Figure 16 Prediction Accuracy SVM

According to figure 16 it can be seen that count vectorizer with stop words is giving the best accuracy score for the model based on Support vector machine. As it can be seend from this plot stemming the texts in the corpus has reduced the accuracy by roughly 1 %. This could be because of over stemming errors. Overstemming errors occur when the stemmer merges two unrelated words that should not have been merged like ‘experience’ and ‘experiment’. Overstemming is known to reduce the precision and accuracy in models.

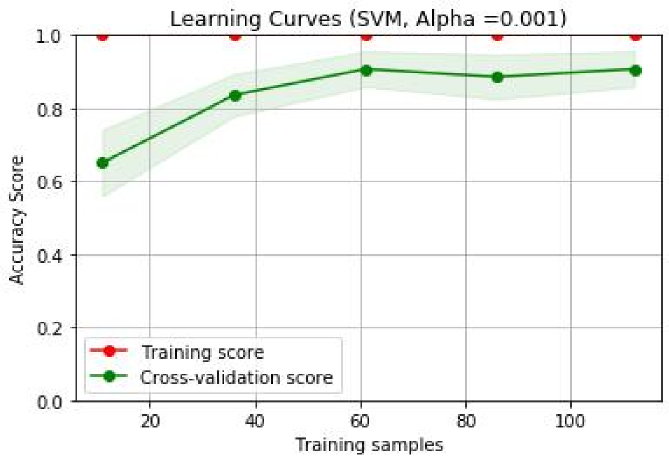


Figure 17 learning curve SVM using count vectorizer and Tf-Idf

Learning curve of model based on SVM and count vectorizer is plotted in figure 17. The figure shows that training score was always close to maximum irrespective of the training samples. However, the cross-validation (cv) score depended on the number of training samples. Comparing figure 15 and 17, it can be seen that SVM model has a much better cv curve than MNB. Also the gap between the curves is less indicating that the dataset has good enough number of training samples in order to give a cross validation score for the model.

Case 3: Model based on K Nearest Neighbor Algorithm

a. Using count vectorizer and Tf-Idf on KNN with 4 neighbors

When K Nearest neighbor algorithm with 4 neighbors (KNN4) is used on the test dataset having the same 114 files (as in previous case studies) using count vectorizer and Tf-Idf the accuracy achieved is 0.8771929824561403. This is lower than MNB and SVM models. Table 5 shows the performance matrix obtained for this model using KNN4, count vectorizer and Tf-Idf.

	precision	recall	f1-score	support
Erroneous	0.78	0.78	0.78	32
Normal	0.91	0.91	0.91	82
Avg/Total	0.88	0.88	0.88	114

Table 5 Performance Matrix – KNN4– using Count Vectorizer and Tf-Idf

b. Using count vectorizer and Tf-Idf on KNN with 6 neighbors

When K Nearest neighbor algorithm with 6 neighbors (KNN6) is used on the same test dataset using count vectorizer and Tf-Idf the accuracy achieved is 0.868421052631579. Thus it is seen that accuracy decreases when number of neighbors is changed to 6 from 4. Table 6 shows the performance matrix obtained for this model using KNN6, count vectorizer and Tf-Idf.

	precision	recall	f1-score	support
Erroneous	0.81	0.69	0.75	32
Normal	0.89	0.94	0.91	82
Avg/Total	0.87	0.87	0.87	114

Table 6 Performance Matrix - KNN6 -using Count Vectorizer and Tf-Idf

From the performance matrix in table 5 and 6 it can be observed that model is making more mistakes in prediction of erroneous logs as the accuracy is low (75%) in erroneous category. This model appears to be not performing so well as per the expectation.

c. Using count vectorizer with stop word removal and Tf-Idf on KNN4

This case is conducted to study the effect of stop words removal along with count vectorizer and Tf-Idf. It is found that stop words removal has reduced the accuracy to 0.8508771929824561. This is somewhat different from the expectation as in all the previous cases stop words removal had helped in increasing the accuracy.

d. Using stemming vectorizer and Tf-Idf on KNN4

This study uses Stemming Count Vectorizer on this model. Interestingly, stemming does not increase accuracy much for this model. Accuracy slightly increases to 0.8596491228070176. Hence it can be assumed that not every model gets benefitted from stop words removal and stemming.

e. Using hashing vectorizer and Tf-Idf on KNN4

Finally the model is tested with hashing vectorizer. The Accuracy of model using KNN4 and hashing vectorizer is 0.8516462482194533. Even hashing vectorizer did not help this model to increase accuracy. The performance matrix using Hashing vectorizer is given by table 7.

	precision	recall	f1-score	support
Erroneous	0.70	0.81	0.75	32
Normal	0.92	0.87	0.89	82
Avg/Total	0.86	0.85	0.85	114

Table 7 Performance matrix -KNN4 – using Hashing vectorizer and Tf-Idf

The line plot in figure 18 depicts how the prediction accuracy of the model changed using different vectorizers.

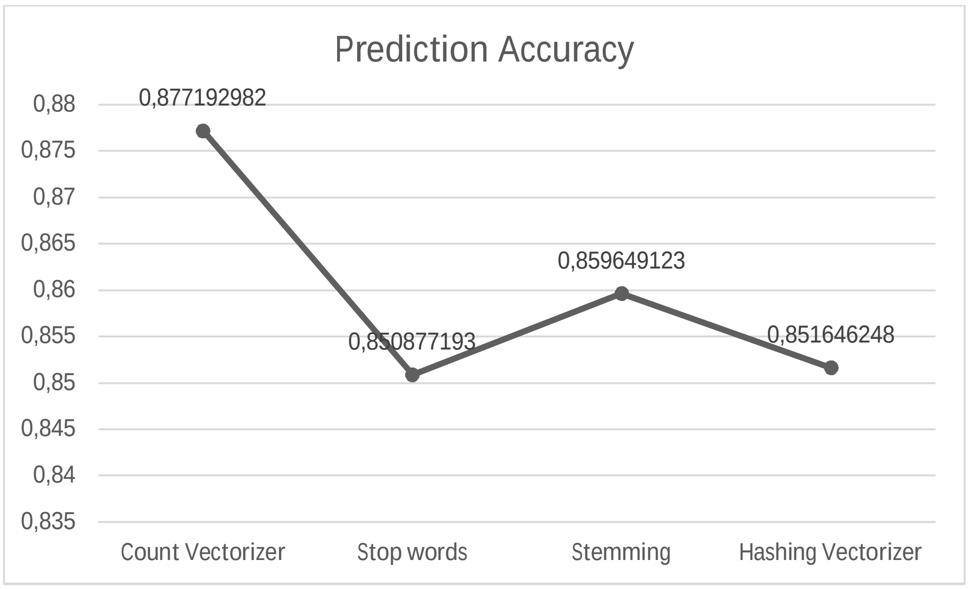


Figure 18 Prediction Accuracy KNN4

According to figure 18 it can be seen that simple count vectorizer with Tf-Idf gave the best accuracy score than by using other methods.

Learning curve of model based on KNN4 and count vectorizer is plotted in figure 19. The figure shows that training score is slightly more than cross validation score, but the plots correspond to each other well and it can be seen that as the training samples increases both the scores increase.

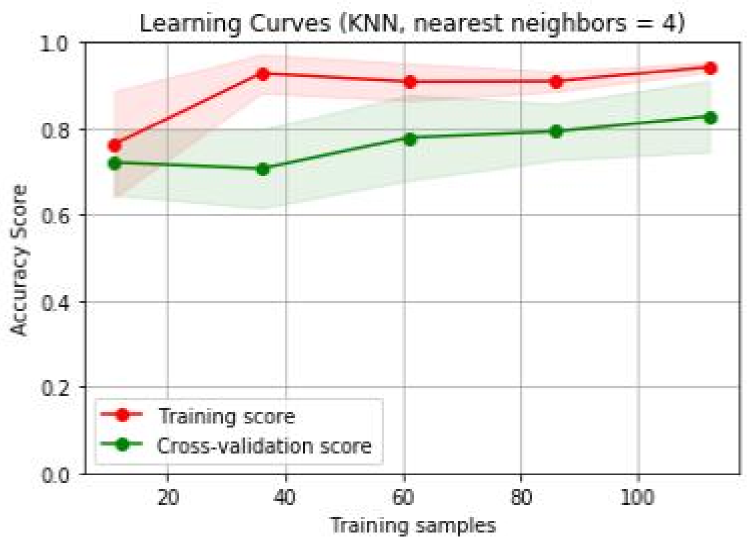


Figure 19 Learning curve KNN4 using count vectorizer and Tf-Idf

Figure 20 shows the variation of prediction accuracy when value of K is changed. It can be seen that accuracy is best when K is 2 or 4. But too low values of K like 1 and 2 can cause an overfitting model. This is because granularity is so fine with low values of K that there will be more outliers in the decision process. Hence to avoid overfitting problem most of the case study was done setting 'K' as 4 and 6. Even then as it has been seen in these case studies, KNN does not seem to be a good choice for the current prototype.

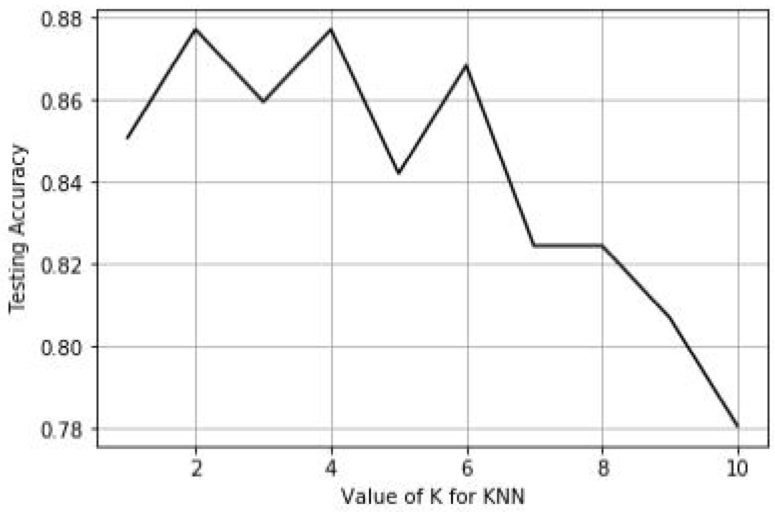


Figure 20 Prediction accuracy Vs Value of K for KNN

Case 4: Model based on Random Forest (RF) Algorithm

a. Using count vectorizer and Tf-Idf

When Random Forest algorithm is used on the test dataset using count vectorizer with Tf-Idf, the accuracy achieved is <u>0.8157894736842105</u>. This is the lowest score achieved for count vectorizer when comapred with all the other models in this case study. Table 8 shows the performance matrix obtained for this model using RF, count vectorizer and Tf-Idf.

	precision	recall	f1-score	support
Erroneous	0.87	0.41	0.55	32
Normal	0.81	0.98	0.88	82
Avg/Total	0.82	0.82	0.79	114

Table 8 Performance matrix -RF- using Count Vectorizer and Tf-Idf

b. Using count vectorizer with stop word removal and Tf-Idf

This case is conducted to study the effect of stop words removal along with count vectorizer and Tf-Idf. It is found that stop words removal has decreased the accuracy to <u>0.6754385964912281</u>. General understanding is stop words should help the model to increase the efficiency. This result could be a side effect of using a 'global' stop word list available in scikit learn for English language rather than making a custom stop-word list made specifically for EC logging context.

c. Using stemming vectorizer and Tf-Idf

This study uses Stemming Count Vectorizer on this model. Stemming has made a good increase in the prediction accuracy by increasing it to <u>0.7543859649122807</u>. Although this is not a very high accuracy score, it can be said that the model has still improved from where it was using stop words and Tf-Idf.

d. Using hashing vectorizer and Tf-Idf

Finally the model is tested with hashing vectorizer. The Accuracy of model using RF and Hashing vectorizer with Tf-Idf is <u>0.7456140350877193</u> which is very close to case 'c' for RF.

Since hashing took a longer time for training and testing and did not increase the accuracy, it can be said that hashing with random forest model should not be used for this prototype. The performance matrix using hashing vectorizer is given by table 9.

	precision	recall	f1-score	support
Erroneous	0.67	0.19	0.29	32
Normal	0.75	0.96	0.84	82
Avg/Total	0.73	0.75	0.69	114

Table 9 Performance matrix-RF-using Hashing vectorizer and Tf-Idf

What is interesting to note in table 9 is the extremely low values of recall and f1-score for erroneous category. This definitely is an indicator that RF with hashing vectorizer must not be used for anomaly detection as there are chances that a faulty log is missed to be categorized as faulty quiet frequently.

The line plot in figure 21 depicts how the prediction accuracy of the model changed using different vectorizers.

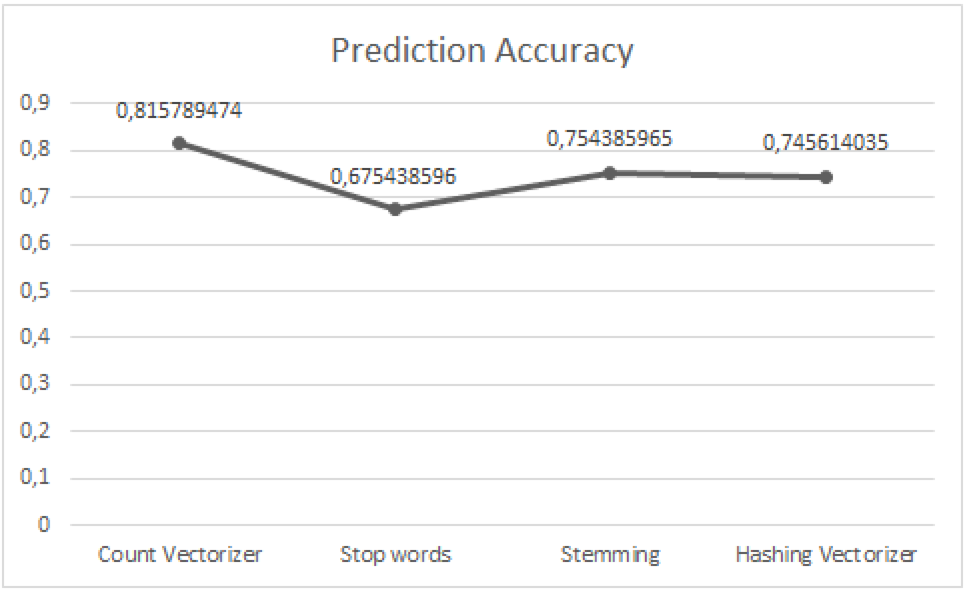


Figure 21 Prediction Accuracy - RF

According to figure 21 it can be seen that simple count vectorizer gives the best score for this model. However while comparing with other models RF turns out to be the least performing model.

Learning curve of model based on RF and count vectorizer is plotted in figure 22. The figure shows that training score was always close to maximum irrespective of the training samples. However, the cross-validation (cv) score starts very low (58 %) and then increases very gradually close to 80 % when the number of training samples is increased.The gap between the curves imply that for this model to perform better it needs more testdata.

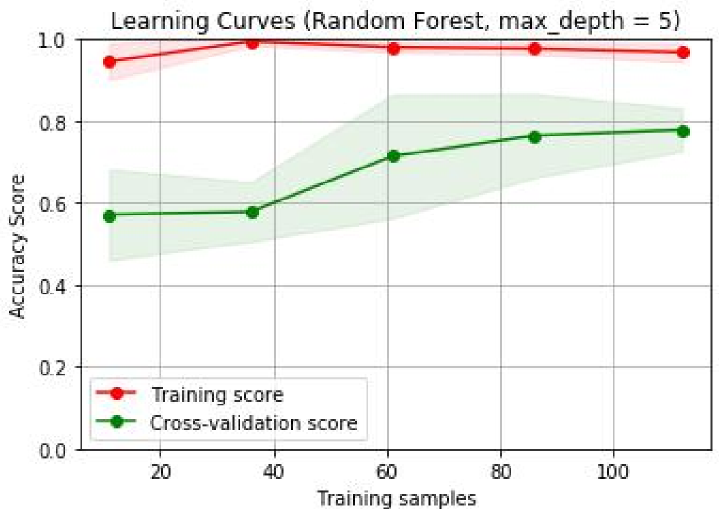


Figure 22 . Learning curve RF

Algorithm performance comparison plots and discussion

a) Classification report Comparison for all models

Most of the times a machine learning model is evaluated based on its accuracy alone. But recall, precision and f1 score are also equally important measures which gives a good overview of any algorithms overall performance. The classification report of any model is the summary of it precision, recall and accuracy on a test dataset. Figure 23 is classification report comparison plot of different algorithms used in the study. The classification report was generated when the model used count vectorizer and Tf-Idf along with the different algorithms. From figure 23 it is evident that Support Vector Machine (svm) is clearly the best performing algorithm when in terms of precision, recall and f1-score.

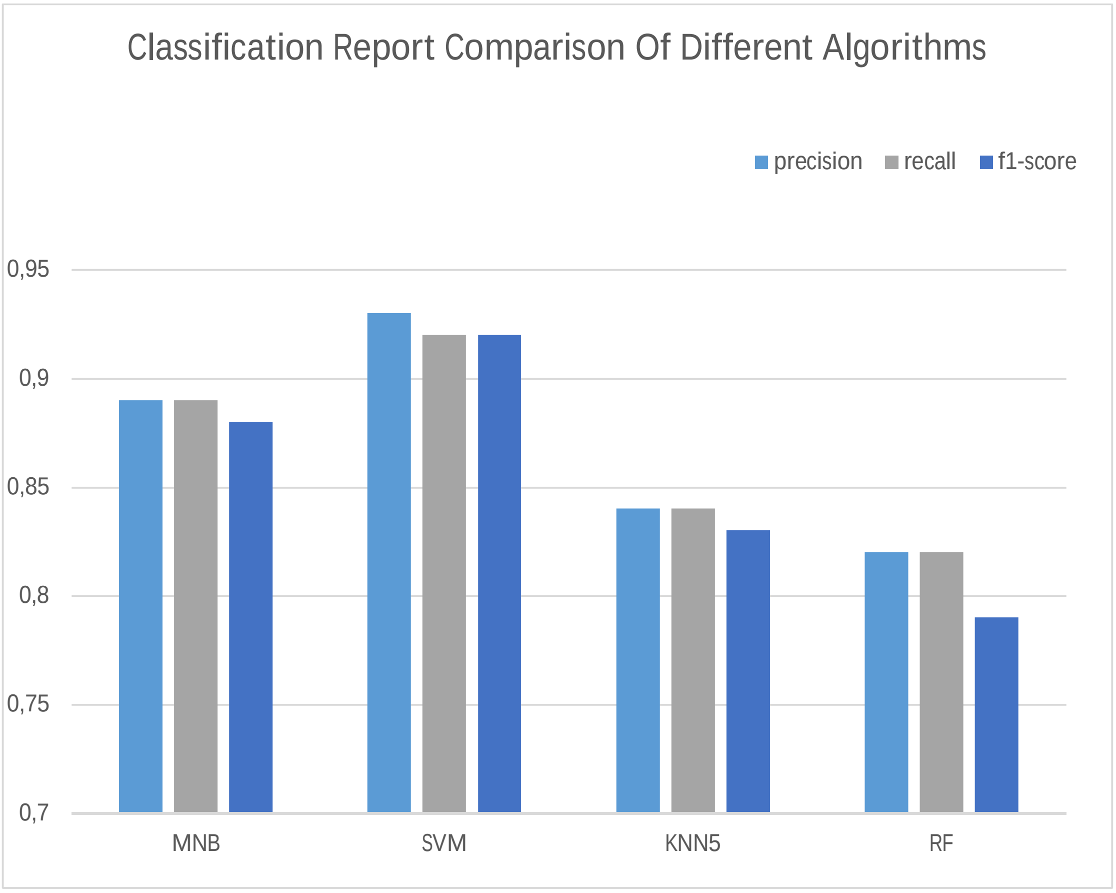


Figure 23 Classification report comparison

b) Comparison of tuning methodology on prediction accuracy for all models

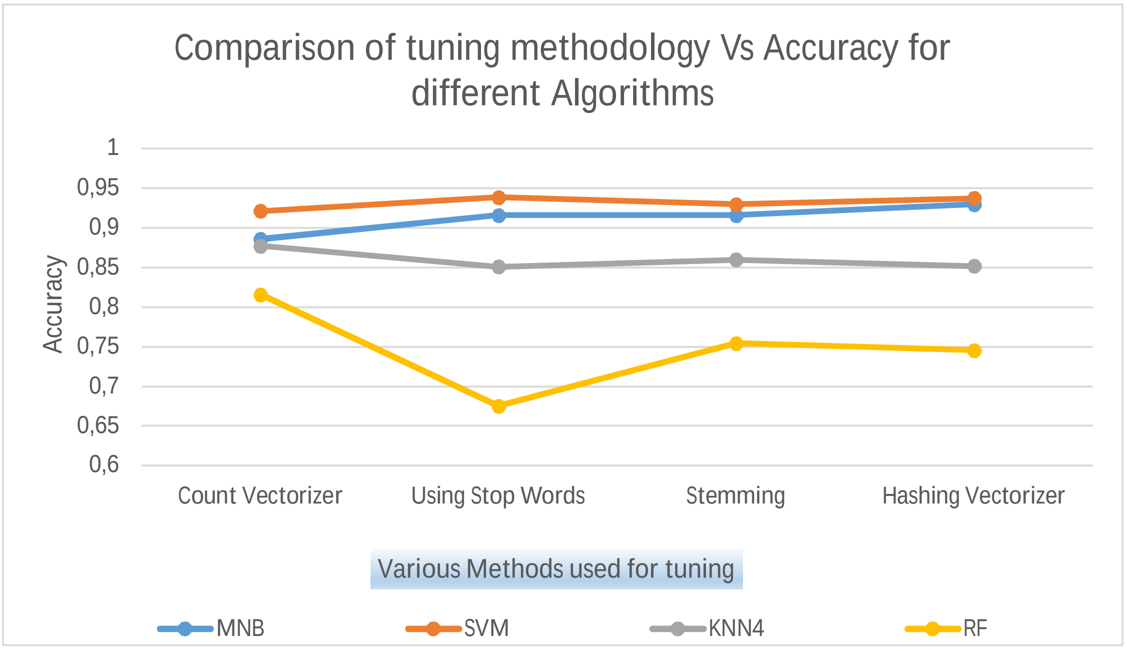


Figure 24 Comparison of tuning methodology

Figure 24 shows the effect of different methodologies (x axis) on the accuracy of the model. As it can be seen both the models based on Multinomial Naive Bayes and Support Vector Machine show the best results for hashing vectorizers. Model based on Random Forest shows the lowest performance and it's the accuracy goes really low with the use of stop words removal. It can also be seen that SVM is very stable and the accuracy is not affected very much by all these methodologies.

c) Algorithm comparison based on the K fold cross validation score where K is 5

K fold cross validation is an evaluation technique for machine learning models which is based on resampling principle. The parameter k here denotes the number of groups that a given dataset is to be divided into. For this study the value of k chosen is 5. What happens here first is after the dataset is split to 5 groups. Later for each unique group, the group is taken as test dataset and rest of the groups are taken as training dataset and fitting is performed on training dataset and model is evaluated against the test dataset to get a cross validation score. This is called one fold. This procedure is repeated many times and a cross validation score is given to the model based on the summary or mean of the scores for each fold. Figure 25 shows the algorithm comparison based on 5-fold cross validation.

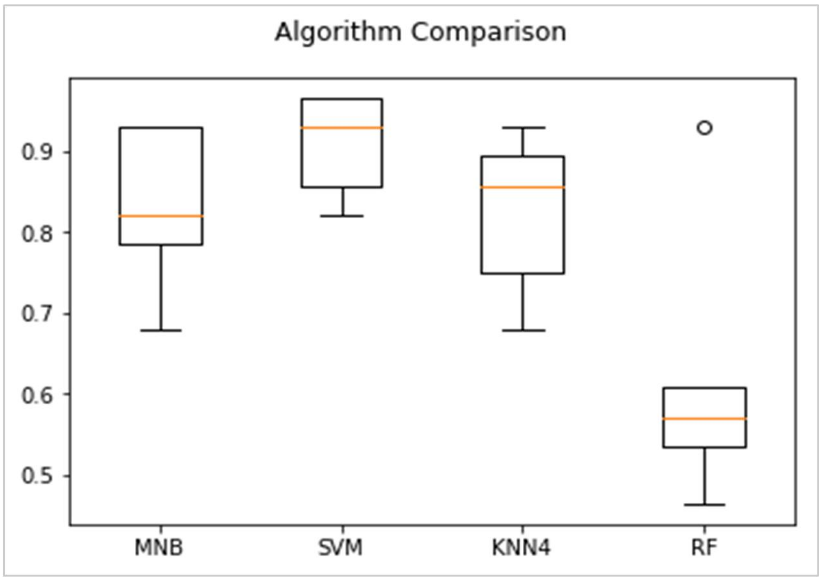


Figure 25 K-Fold cross validation plot

The box plot in figure 25 shows the mean and standard deviation of prediction accuracy scores for each algorithm. Each box shows the range of the accuracy scores across each fold of cross validation for each algorithm. As seen in the plot, SVM gives the highest mean accuracy and lowest standard deviation accuracy as given in table 10. Also it is worth mentioning that although MNB did not give a very good mean, the variance is still very low. Hence it can be considered as an option to be used in this prototype, provided mean can be increased using good tuning methods.

	Mean	Standard Deviation
MNB	0.828571	0.094221
SVM	0.907143	0.058029
KNN4	0.821429	0.093131
RF	0.621429	0.160675

Table 10 Mean and Standard deviation for the different algorithms.

d) **Algorithm comparison based on the training time , testing time and score**

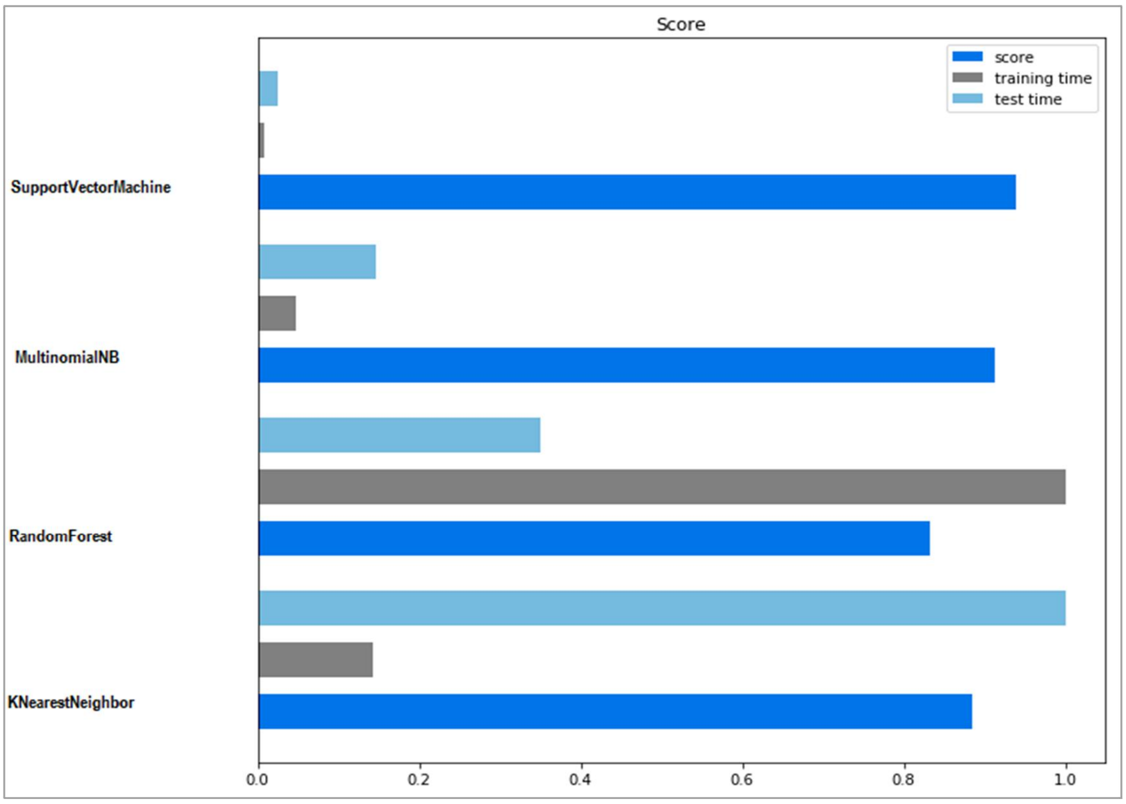


Figure 26 comparison of training, testing time and score

In the comparison done in figure 26, the training and testing time for models based on all the different algorithms used in this study. These timings are then plotted along with the f1 score of the model. As it can be seen from the plot model based on Random Forest algorithm takes the highest time to train but does not give really good f1 score. This once again proves that in the current prototype, Random forest algorithm will turn out to be most expensive in terms of resources and false prediction rate.

K nearest neighbor algorithm with 4 neighbors takes not much time to train the model, but a very long time to test it. Moreover, the f1 score of this model is not very good. Hence it can be concluded that Knn is not a suitable algorithm for the current prototype.

Summary

From the above comparisons and case studies, it can be summarized that for the current problem, Support Vector Machine algorithm performs best in terms of learning time, testing time, precision, f1 score, recall and accuracy. In a model that uses hashing vectorizer along with support vector machine algorithm, a high accuracy of 93 % is achieved.

Conclusion

We are living in an era where logs are extensively used for detection of anomalies in large-scale systems. On the other hand, there is still a lot of dependency on manual log analysis which is becoming increasingly cumbersome due to increasing sizes of log files dumped by these large-scale systems. In order to bring down manual effort, there has been few studies conducted on the topic of automated log analysis and anomaly detection methods in recent years.

In this thesis such a prototype of a tool is developed which can parse the unstructured Tieto EC server logs and analyze these logs automatically based on the features extracted from the contents and supervised learning algorithms. The prototype is also equipped to send an email alerts if any faulty logs are found. Additionally, a toolkit of this anomaly detection model is provided in Tieto repository for reuse and further study. Many case studies, using variants for 4 machine learning algorithm with different tuning parameters are performed and results are analyzed using various performance metrics.

On the basis of these case studies it has been found that a reliable and stable health check analyzer can be made with ELK stack and scikit learn. The experiment results showed that SVM algorithm with parameter alpha = 0.001 shows the best accuracy result among the other compared algorithms. A high performance rate with 92% precision and 93 % accuracy is achieved by using SVM algorithm in the model. It was also seen that Multinomial Naive Bayes also gave a comparatively satisfactory result. The least performing model was when Random Forest Algorithm was used.

This paper also presents the training and testing time for each of these algorithms, the k-fold cross validation results, performance matrix and learning curves so that a clear best performing algorithm can be easily marked. Clearly enough SVM algorithm stand out as the best performing algorithm for the problem in this thesis. All the performance measures showed good correlation between prediction accuracy and number of samples.

Future work

Based on the present study it would be interesting to see how SVM algorithm reacts to a much larger log files dataset. The main hurdle for this is that generation of different scenario based logs often take a lot of time as this involves a lot of configuration in a specific way to make things fail or not fail. Hence dataset generation part for this study was very time consuming and exhausting task.

Another interesting thing to try out is to see if the performance gets better or worse when stack trace in the log file messages is removed. The main assumption for not removing it in this study was that stack traces sometimes contain specific words that can be pointing towards a faulty log. Hence it was considered beneficial for classification. One possible extension to this work would be to write the result back to Elasticsearch in some manner. Thus we can use Kibana to further visualize the results. Also there are multiple plugins for Elastic search like – Elastalert, watcher etc. which can be used to alert in a very secure and controlled manner.

Another possible future work would be to make this prototype also predict based on time series data. This would mean based on events occurring at different timestamps, it would be able to make a prediction if an anomaly can occur in future or not. But this is easier said than done, because not always same events happen before a particular type of failure, and chances of this model predicting incorrectly regarding an anomaly could be higher. Nevertheless, this is still an area interesting to explore.

References

[1] Hydrocarbon accounting with Energy Components, 2017, Can be accessed at https://www.tieto.com/industries/oil-and-gas/energy-components.

[2] Weixi Li, 2013. Automatic Log Analysis using Machine Learning: Awesome Automatic Log Analysis version 2.0.

[3] Wiley, 2011. A.Webb and K. Copsey, Statistical Pattern Recognition.

[4] C. C. Aggarwal, 2005. On abnormality detection in spuriously populated data streams, In Proceedings of ACM SIAM Conference on Data Mining.

[5] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, 2009, Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, pp. 609_616.

[6] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and
Y. L. Cun, _Learning convolutional feature hierarchies for visual recognition.

[7] C. Lagoze, 2003, NSF DL position paper. In NSF Post Digital Library Futures Workshop.

[8] "Data Mining Curriculum". ACM SIGKDD. 2006-04-30.

[9] Hay, A., Cid, D. and Bray, R. (2008). OSSEC host-based intrusion detection guide,
Syngress Pub, Burlington, Mass.

[10] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. (2009, Oct.). Detecting Large-Scale System Problems by Mining Console Logs.

[11] H. Chen, 2003. Towards building digital library as an institution of knowledge. In NSF Post Digital Library Futures Workshop.

[12] T. M. Mitchell, 1997, "Machine learning. 1997," Burr Ridge, IL: McGraw Hill.

[13] Chuvakin, 2013, Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management, Syngress, Waltham,
Mass.

[14] Stuart J. Russell, Peter Norvig (2010) Artificial Intelligence: A Modern Approach, Third Edition, Prentice Hall ISBN

[15] Hodge, V. J. and Austin, (2004), `A Survey of Outlier Detection Methodologies',
Artificial Intelligence Review 22(1969), 85-126.

[16] Cuong Nguyen, Yong Wang, Ha Nam Nguyen, 2013.Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic.

[17] Aboullaite Mohammed, (2017), Docker monitoring with ELK stack, https://aboullaite.me/docker-monitoring-with-the-elk-stack/

[18] D. Jurafsky and J. Martin, 2000, Speech & Language Processing. Pearson Education India, 2000

[19] ELK Stack documentation, can be accessed at https://www.elastic.co/elk-stack

[20] Chen, M.-S., Han, J. and Yu, P. S., 1996, `Data mining: an overview from a database perspective', Knowledge and data Engineering, IEEE Transactions on 8(6), 866-883.

[21] Chapman and Hall, 2010, Data Mining and Knowledge Discovery.

[22] Jacobs, A. (2009), `The pathologies of big data', Communications of the ACM 52(8), 36-44.

[23] Marr, Bernard. (2014), ‘Big Data: The 5 Vs everyone must know’

[24] W. B. Cavnar and J. M. Trenkle, 1994, “N-gram-based text categorization,” Ann Arbor MI, vol. 48113, no. 2

[25] Daniel Berman, last updated on April 2 2018. The Complete guide to ELK stack-2018, Can be accesses at https://logz.io/learn/complete-guide-elk-stack/

[26] The Stanford Natural Language Processing Group (2015), `Stanford tokenizer'. Can be accessed at http: //nlp.stanford.edu/software/tokenizer.shtml/. Last accessed: May 12, 2015

[27] Elastic Documentation, can be accesses at https://www.elastic.co/guide

[28] Apache Software Foundation. (2016), Apache Lucene. Apache LuceneTM 5.5.0 Documentation.

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

[30] Lee, W., Stolfo, S. J. and Mok, K. U. I. W. (2001), `Adaptive Intrusion Detection: A Data Mining Approach', pp. 533-567.

[31] Patrick Kleindienst, (2017). Building a real-world logging infrastructure with Logstash, Elasticsearch and Kibana.

[32] Fred L. Drake, Jr., 1990-2018, Python Software Foundation.

[33] Jaison Brownlee, 2017, Natural Language Processing.

[34] T. Sipola, A. Juvonen, and J. Lehtonen, 2011, “Anomaly Detection from Network Logs Using Diffusion Maps,” Engineering Applications of Neural Networks.

[35] Scikit Learn Documentation can be accessed at http://scikit-learn.org/stable/documentation.html

[36] Dasa Munkova , Michal Munk , Martin , 2013 , Influence of Stop-Words Removal on Sequence Patterns Identification within Comparable Corpora, Part of Advances in Intelligent System and Computing book series (AISC, volume 231)

[37] Vasilis Vryniotis, 2013, The Naive Bayes Text Classifier

[38] Renuka Joshi, 2016, How to evaluate the performance of a model in Azure ML and understanding “Confusion Metrics”

[39] Elastic search API documentation, can be accessed at http://elasticsearch-py.readthedocs.io/en/master/api.html.

[40] Hwanjo Yu and Sungchul Kim, 2013, SVM Tutorial: Classification, Regression, and Ranking

[41] ML bot1, 2017, Understanding Support Vector Machines: A Primer

[42] Support vector machines: The linearly separable case can be accessed at https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html

[43] Ho, Tin Kam, (1995). Random Decision Forest. Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.

[44] Hastie, Trevor , Tibshirani , Robert , Friedman, Jerome ,(2008). The Elements of Statistical Learning (2nd ed.) page -587-289.

[45] K Nearest neighbors can be accessed at https://nlp.stanford.edu/IR-book/html/htmledition/k-nearest-neighbor-1.html