Anomaly Detection in HDFS Log Files

Disha DH (dishaholla04@gmail.com), Neethu G (neethu.g0708@gmail.com),
University of British Columbia, Master of Data Science, Canada

Abstract*—As the technologies are progressing, the data generated from these technologies also keeps increasing. Processing the data and extracting the key information from it becomes of key importance with the growing amount of data. The world of data is so large that for a minute it is absurd to think that even a single click generates an event in the background which is considered as a log message. Log files generally contain all the events associated with a file or a system. These log files are usually kept, to keep track of what is happening in the background and in case of malfunctioning of a system, we can have access to these to check what exactly went wrong. The next important aspect of managing the log files is anomaly detection within the log files. Manually checking for each incoming log and concluding if a log is an anomaly or not is next to impossible considering the number of logs generated by a single system. This project will aim on automating this process of detection of anomalies in log events. This will mainly target Hadoop file system logs. Hadoop Distributed File System is a distributed file system which is highly fault tolerant. It handles large datasets running on commodity hardware. This project will aim on parsing the HDFS log file to fit machine learning models with the highest accuracy to test if any incoming log file is an anomaly or not. The HDFS log file is first parsed using basic tools like excel and python to make it into a readable format post which the data is parsed and wrangled to keep only the required data. With the use of feature extraction technique, each log message is parsed to gather the count of the number of repeated words for each event. This data is then passed to machine learning algorithms with training, test and validation sets to develop a model with low accuracy which predicts if an incoming log message is an anomaly or not based on the training data provided to the model. The events are grouped based on block ids and the time frame to group the events within the same second of time. This step is done to ensure there are no multiple events within a second which could be an anomaly like a Bruteforce attack. The parsed and grouped data is then trained and modeled using random forest. Feature extraction technique is used to know the importance of each variable and the unimportant variables are dropped and the model is fit again to improve accuracy.

Index Terms—Anomaly, Log files, Feature extraction, Feature importance, Classification, HDFS.

I. Introduction

Data analysis mainly came into picture because of the immense amount of data that is produced by the various systems we use. Managing this large amount of data becomes of key importance especially to big multinational companies like Google and Facebook which generate trillions of GB of data in every one second. Hadoop is this system which manages the big data. HDFS is a storage system for Hadoop. The HDFS has a "namenode" and multiple "data nodes" which are simultaneously used for storing the data in such a way that there is faster access to the data. Monitoring the logs to check for any anomaly should be considered of utmost importance as these systems store a lot of data and crashing of even one of these systems can be of a great loss.

1.1 Purpose:

The purpose of this project is to develop a model which analyses the logs of an HDFS to predict if an incoming log is an anomaly or not. The input log fed to the model contains labeled tags to classify the events as an anomaly or not. Tools like excel and python programming are used for log analysis, feature extraction and data wrangling. The incoming log file is parsed using python and excel to extract the count of a few important words in the log message and the weight of these words is calculated for each word in the sentence.The data is then parsed again to keep only the desired columns and is divided into training and testing set. Using the training set, the ranger function in R is used for fitting the random forest model and the inbuilt feature importance method is used to drop the columns. This project can be used as a method to check if an incoming log message is an anomaly or not based on the Block ids, words in the log message and the time stamp of the log.

II. Literature Review

The fundamental aim of this project is to experiment with methods of anomaly detection applied to log files. Anomaly detection methods are daily used in a variety of domains across industries. Anomaly detection methods are instrumental in the banking sector, protecting us against fraudulent activity and transactions. In the cybersecurity domain, these methods help researchers to detect attackers' sophisticated malicious software. The methods also help scientists and doctors to detect cancer or chronic illness in the healthcare industry. Last but not least, anomaly detection methods monitor the behavior of a distributed system in a data center and detect an anomalous state of a particular machine, which is the scope of this thesis.

With the current demand for cloud services, it has become increasingly important to provide highly available and error-free services. High availability of services can be achieved by quickly identifying the root cause. Furthermore, log files are one of the possible solutions. Nowadays, no company can afford to inspect the log files manually. Therefore, there is a surge of interest in developing methods which can automatically detect anomalous behavior with high accuracy.

2.1 Categories of anomalies:

1. Point anomalies:

We say that a particular data point is anomalous if it significantly deviates from the rest of the data points

2. Contextual anomalies:

A particular data point belongs to this category if it is anomalous only in a specific context but not otherwise. There are contextual and behavioral attributes that define each data point. Contextual attributes determine the context or the neighborhood of the data point. In contrast, behavioral attributes define the non contextual characteristics of the data point. We often observe contextual anomalies in time-series data such as temperature, stocks, and logs

3. Collective anomalies:

A collection of data points is a collective anomaly if it is anomalous with respect to the entire data set. Individual data points are not usually anomalies, but their occurrences together represent an example of a collective anomaly.

The common structure of a log file is depicted below.The log statement usually corresponds to a single line of text in a particular log file. The log statement comprises two parts: a log header and a log message. The log header records information about a timestamp, a level of severity, or a process identification number (PID).Since the same formatting is used for all types of logs, they are relatively easy to parse. The log message is further divided into a log key and log parameters. The log key is a static part that is the same for all log lines corresponding to a particular event. On the other hand, the log parameters are variables that may vary within one event. A representative example of log parameters is an IP address or the size of a file.

```
logging.info(f 'PacketResponder {packet_id} for block {block_id} terminating')
```

```
081109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block
blk_38865049064139660 terminating
```

Header	Timestamp	081109 203615
	PID	148
	Level	INFO
	Component	dfs.DataNode$PacketResponder
Message	Key	PacketResponder ? for block ? terminating
	Parameters	1, blk_38865049064139660

Fig 1. Sample of Log file

2.2 Approaches for log parsing

Traditionally, a log anomaly detection approach comprises log collection, log pars- ing, feature extraction, and anomaly detection.

Log collection:

Collecting log files is a crucial feature of a very large scale system.Nowadays we have all applications which generate log files at the back end.

Log Parsing:

As mentioned above, log files contain semi-structured data, which should help ad- ministrators with a root cause analysis. The goal of log parsing is to transform semi-structured data, i.e., log files, into non-specific structured data. We used the oldest approach based on handcrafted regular expressions, which extract log keys and log parameters

Feature extraction:

Extracting the most appropriate but still uncorrelated features is an essential subtask in every machine learning project.

Anomaly detection:

Once the required features are extracted we then use those features to obtain the weight of these features from the log message to fit a model and detect anomalies.

III. WORKFLOW

Data is a very powerful tool which can be used in various technologies to predict things that can happen in future. Statistics has always ruled over various industries for data reasoning. Statistics combined with modern day technologies can act as an extremely powerful tool in automatizing various things which could take a human several hours. As the technology is growing, each technology also produces a large amount of data. Social media for example generates billions of GB of data every day. This data can be of much importance to the companies as this can be used as a powerful medium for advertising and feature pooling. All these major data producing companies rely on the cloud for storage. The HDFS is the Hadoop file system storage used to store file system logs on the cloud. The HDFS stores and processes billions of log messages. Manually parsing these logs to detect any anomaly can be a very tough job as each data has a completely unique data. For example the timestamp cannot be the same ever. This project aims at automatizing this process of anomaly detection using the below workflow:

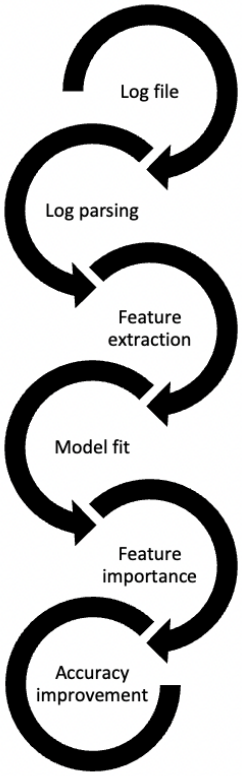

Fig 2 . Workflow

3.1 Log file:

The HDFS dataset considered is in the .log format and has all the data in form of texts. This log file is generated using the benchmark loads on a private cloud environment. The logs are divided into traces according to the block ids. Each Block ID is then assigned with a response variable as "Anomaly" or "Normal". The data is composed of two files:

- ➔ Anomalylabel.csv: This file consists of the block ids of each log message adjacent to which is a column stating if the block id is an Anomaly or not. There are 575,062 rows in this file out of which 16,838 are anomalies and the rest are categorized as normal.
- ➔ HDFS.log: This file consists of log messages in raw format. This project has used excel and python programming language to parse this data into readable format.

Dataset:t https://zenodo.org/record/3227177#.Yl24B PMK

3.2 Log Parsing:

The HDFS log file is read using excel and is further parsed to separate the data to the respective columns. After this data is read in excel, the following columns are found in the data:

- ➔ **Date** : The date at which the log was created(YYYY/MM/DD)
- ➔ **Times** : Time the log was generated(HH:MM:SS)
- ➔ **PID** : The process ID of the log.
- ➔ **Log Component** : The information the log is conveying
- ➔ **Log Message** : The message of the log.

The total number of rows in the file are 10,48,575. The parsed file looks like the one shown in Figure 3.

The log message was further parsed to extract the block id from the log message. Each block id was further compared with the block id column in the Anomalylabel.csv file using the following command:

```
=VLOOKUP(B12; A2:C10; 3; 1)
```

Date	Time	PID	Log component	Log Message
08/11/09	8:35:18 PM	143	INFOdfs.DataNode$DataXceiver:	Receivingblockblk_-160899968791986290 6src:/10.250.19.102:54106dest:/10.250.19.102:50010
08/11/09	8:35:18 PM	35	INFOdfs.FSNamesystem:	BLOCK*NameSystem.allocateBlock:/mnt/hadoop/mapred/system/job_200811092030_0001/job.jar.blk_-1608999687919862906
08/11/09	8:35:19 PM	143	INFOdfs.DataNode$DataXceiver:	Receivingblockblk_-160899968791986290 6src:/10.250.10.6:40524dest:/10.250.10.6:50010

Fig 3. Parsed log file

The parsed excel file now had block id corresponding to each log message and each log message had a corresponding response variable stating if the log was an anomaly or not.

Each log component was parsed further using the python command to extract the IP address(source and destination), port number(source and destination, and file size.

Example code to extract destination IP address:

```
data['DestinationIP']=data['Log
Message'].str.extract('(dest./\d{1,3}\.\d{1
,3}\.\d{1,3}\.\d{1,3}.\d{1,5})',
expand=True)
```

- On extracting these columns and taking the uniqueness of each column, it was found that all the IPs were internal IPs. Hence there was no much significance considering these IPs as there cannot be a Bruteforce attack considering only internal IPs. These IPs exchange messages every minute and every log message exchange can happen within minutes. Creating a rule for a Bruteforce attack would generate multiple False positive cases here considering all the internal IPs.
- The source ports were in the range 32771 to 60999. This is the port used by the communicating system to send the message. Using malicious ports for communication can be considered an anomaly. But in this case since all the source IPs were internal IPs, the ports were all found to be safe.
- There was only one destination port that is 50010 which is the default HDFS port used for file transfer. Hence this also did not show much importance in anomaly detection.
- The size component range was from 0-99989. All the logs which had the size variable as 99989 were also found to be having Normal response variable. This variable only mentioned the size of the file exchanged during the data transfer. It cannot be considered that if a larger file is exchanged the log is an anomaly.

3.3 Feature extraction:

In this step, the excel file data was used further to count the occurrences of each word in the log message. 33 words were considered. Further the weight of each word was calculated by dividing the number of occurrences of the word in the log message with the word count in the log message.

Find total number of words in a log message:

```
=IF(LEN(TRIM(A2))=0,0,LEN(TRIM(A2))-LEN(SUB
STITUTE(A2," ",""))+1)
```

Find the count of each word in a log message:

```
(LEN(B5)-LEN(SUBSTITUTE(UPPER(B5),UPPER(C5)
,"")))/LEN(C5)
```

The unique words were taken using the find unique command in excel and the words were considered in each log message. The count of occurrences of each word was divided with the total word count in the log message to find the weight of each word in the log.

3.4 Model fit:
Once the excel file was ready with the required fields that is the Date and time, PID, Block ID, weights of each word in the log message, the file was converted into a csv and imported in R studio.

- The date and time columns were combined into a single column and converted into the right format.
  Example code for date time conversion:

```
data$date<-paste(data$Date,"
",data$Time)
data[['date']]<-as.POSIXct(data[['dat
e']],
format = "%Y-%m-%d %H:%M:%S")
```

- Further the rows containing the response variable as NA were eliminated and the response variable was factorized.
  Example code to remove NAs in response column:

```
data<-data[!grepl("#N/A",
data$Response),]
```

- The data was divided into training and testing sets with a 70 - 30 split.
  Example code for dividing the data:

```
dt = sort(sample(nrow(gdata),
nrow(gdata)*.7))
gdata.train <- gdata[dt, ]
gdata.test <- gdata[-dt, ]
```

- The response variable was dropped from the test set.

```
Response <- gdata.test$Response
gdata.test <-subset(gdata.test,
select=-Response)
```

- Random forest model was fitted on the training set using the 'ranger' function in R.

  Example code to fit model:

```
install.packages("ranger")
library(ranger)
rf <- ranger(Response ~ ., data =
gdata.train, write.forest =
TRUE,mtry=3, importance = 'impurity')
```

- The confusion matrix of the training set gave the training accuracy of the model.
  Training set confusion matrix:

True Vs Predicted	0(Normal)	1(Anomaly)
0(Normal)	562288	0
1(Anomaly)	9882	135435

Table 1: Train 1 confusion matrix

Training accuracy:
(1-(9882/(9882+135435+562288)))*100 = 98.60%

- Predict response of the test set using the random forest model:

```
pred <- predict(rf, data =
gdata.test)
```

Confusion matrix for test set:

True Vs Predicted	0(Normal)	1(Anomaly)
0(Normal)	240878	0
1(Anomaly)	8403	53979

Table 2: Test 1 confusion matrix

Test accuracy:
(1-(8403/(8403+240878+53979)))*100 = 97.22%

- Total number of trees used for the random forest model were- 500.

3.5 Feature importance:
The default feature of the ranger function was used to calculate the importance of each variable in the data.

Example code to calculate feature importance:

```
rf$variable.importance
```

Results of feature importance:

PID	Block_ID	Receiving	Block	Served	terminating	Received
8.751751e+00	1.406047e+05	1.220303e+00	1.186854e+00	6.044290e+00	2.310257e-01	5.625843e-01
root	updated	packet.responder	verification	succeeded	exception	hadoop
8.423906e-02	3.996918e-01	2.397190e-01	1.402613e-01	1.346699e-01	8.361385e-01	2.396936e-01
thread	starting	transfer	transmitted	delete	serving	error
1.075682e-01	8.488112e-02	1.044768e-01	3.669270e-01	2.352424e-01	2.498222e-01	5.675441e-01
unexpected	redundant	request	failed	info	warn	scanner
5.239823e-01	1.011113e-01	1.074777e-01	5.435709e-03	1.160430e+00	5.138042e-01	1.583390e-01
Xceiver	Dataset	Datatransfer	Namesystem	Responder	date	
6.803240e+00	2.796477e-01	3.660183e-01	5.292395e-01	4.703940e-01	1.758589e+01	

Figure 4. Feature importance

The two components with least importance were removed from the data.

Example code to remove least important variable:

```
data2 <- subset(gdata, select=-c(root,succeeded))
```

3.6 Accuracy improvement:

- The new data was further divided into training and testing sets and the random forest model was fit again using the ranger function.

Training set confusion matrix after feature importance:

True Vs Predicted	0(Normal)	1(Anomaly)
0(Normal)	562183	0
1(Anomaly)	15605	129817

Table 3: Train 2 confusion matrix

Training accuracy:
(1-(15605/(15605+562183+129817)))*100=97.7%

Confusion matrix for test set:

True Vs Predicted	0(Normal)	1(Anomaly)
0(Normal)	240983	0
1(Anomaly)	8176	54101

Table 4: Test 2 confusion matrix

Test accuracy:

(1-(8176/(8176+240983+54101)))*100 =97.3%

IV. RESULTS AND DISCUSSIONS

As seen in the previous section, the random forest model attained a test accuracy of 97.2% before feature importance was extracted and further improved accuracy of 97.3% after the feature importance technique was adapted. This is quite a good model to predict the log message anomaly considering the test set accuracy. Given the Block ID, PID, timestamp and the log message, the log can be categorized as an anomaly or not using this model.

V. CONCLUSIONS AND FUTURE ENHANCEMENTS

This model will take in the desired inputs to parse the data and predict if the log is an anomaly or not. This is a very basic approach adapted to construct this model. The model has an acceptable accuracy hence this can be considered to develop better models.

- Various online parsers can be used to parse the data instead of manually parsing the data. This was mainly done manually to understand the data better. Various online parsers like dynatrace can be used to automatize the parsing process and improve the model even more.
- Various visualization techniques can be adapted and dashboards can be created which will be fed live events from the HDFS logs to continuously monitor the events for any anomaly.
- This algorithm can be applied in various SIEM(Security Incident and Event Management) tools in which these logs are provided as an input.
- Various other modeling approaches like neural networks can be adapted to improve the model accuracy.

REFERENCES

[1] https://github.com/logpai/loghub/tree/master/HDFS#hdfs_1

[2] http://hadooptutorial.info/log-analysis-hadoop/

[3]https://www.perfectxl.com/excel-glossary/how-to-use-vlookup-excel/#:~:text=VLOOKUP%20stands%20for%20'Vertical%20Lookup,column%20in%20the%20same%20row.

[4] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[5] Deep Learning-based System Log Analysis for Anomaly Detection-Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, Michael R. Lyu

[6] CausalConvLSTM: Semi-Supervised Log Anomaly Detection Through Sequence Modeling Steven Yen; Melody Moh; Teng-Sheng Moh

[7] Anomaly Detection Methods for Log Files- Bc. Martin Koryt'ak, Ing. Jan Drchal, Ph.D. , Open Informatics, Department of computer Science

[8] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael Jordan. Detecting Large-Scale System Problems by Mining Console Logs, in Proc. of the 22nd ACM Symposium on Operating Systems Principles (SOSP), 2009.

[9] LightLog: A lightweight temporal convolutional network for log anomaly detection on the edge- ZuminWang, JiyuTianaHuiFang, LimingChen, JingQin.