Predicting Anomalous Logs in Hadoop Distributed File System

Nyanda Redwood
Department of Computer Science
The University of British Columbia
Kelowna, Canada
nredwood@gmail.com

***Abstract*—Machine Learning is helpful in many scenarios. In the current case, which pertains to predicting anomalous log files, I use techniques from Machine Learning to compute the prognostication in question. The log files are from Hadoop Distributed File System (HDFS) and are representative of Big Data as their count goes over 11 million rows of data.**

With the increasing prevalence of Big Data comes the increasing need to mine it efficiently for critical pieces of information that can prove beneficial to institutions, companies, and organizations as well as to the public, as witnessed with COVID19.

The forecasting process commenced with the creation of a parser code to re-structure raw log files (HDFS). Throughout, I performed Exploratory Data Analysis to garner insights. In the end, I employed Scikit Learn Dummy and Random Forest Classification functions to perform Supervised Machine Learning on the HDFS log files. The results illustrate the successful training of the computer to correctly label log files as anomalous or as normal. The precision scores are 0.95 and 0.99 with recall at 0.60 and 1, and F1-scores at 0.74, and 0.99 respectfully.

Key terms—Machine Learning, Big Data, Hadoop Distributed File System, Scikit Learn, and Random Forest Classification

I. Introduction

This paper employs machine learning technology to examine one specific category of Big Data, log files. Its aim is to predict anomalous log files. These log files are from the Hadoop Distributed File System (HDFS). HDFS is a "distributed file system designed to run on commodity hardware" [3]. These logs originate from the *GitHub* repository of Log Analytics Powered by AI (*LOGPAI*), and specifically within the branch called *Logpai/Loghub*. The name of these log files on this branch is "HDFS_1". These files form a part of a broader parsing project to which *Logpai/Loghub* is dedicated, and upon which improvements have been made by other parties such as the *Github* repository *IBM/Drain3*. *Logpai/Loghub* contains partial datasets including of HDFS_1, and, so, following *Logpai/Loghub* 's direction, I obtained the full datasets on the webpage *ZENODO* [5][8].

II. Literature Review

This section briefly examines the state of the question as it pertains to log parsing and log mining and analysis. It does by citing two specif articles: "Tools and Benchmarking for Automatic Log Parsing" and "Drain: An Online Log Parsing Approach with Fixed Depth Tree"[4][7]. However, it is informed by additional research beyond these two.

A. "Tools and Benchmarking for Automatic Log Parsing"

In their 2019 article "Tools and Benchmark for Automated Log Parsing", Jiemeng Zhu et al highlight the importance of log parsing and the redundancy of older methodologies in performing this action due to the occurrence of Big Data. In this respect, they bring to light the use of new methodologies such as machine learning tools in the efficient analysis of log files. Key to these new analytical approaches is the advent of automated log parsing. Jiemeng Zhu et al aim is to investigate the use of automated log parsing. To achieve this, they utilized sixteen batches of log files that all fall within the category of Big Data and thirteen automated log parsers. They put forth three markers of success in evaluating the performance of the thirteen automated parsers, which are accuracy, robustness, and efficiency. Their hope is for their work to be used as a key foundation for future works, in both the practical and academic spheres. Along with providing ample inclusion of other academic studies, they were also able to implement their findings on a practical level with Huawei.

In their experiment, they found Drain, one of the thirteen existing automated log parsers at the time, to be the most accurate. Drain also performed well on their robustness metric; however, they found that all thirteen parsers performed poorly on complicated log files with numerous event templates such as the Android dataset. Similarly, the parsers performed inadequately on the metric of efficiency for log files with numerous event templates.

Throughout their article, several claims were made. The first is that log files are central to the successful performance of software systems. This is because they comprise of an historicization of all activities, which proves beneficial in investigating errors. In this regard, they underscore the importance of being able to re-structure unstructured log files

via log parsing to facilitate access to the information stored within log files. In their advocacy for automated log parsing as opposed to manual log parsing, including coding, they reasoned that automation entails the learning of log patterns from which event templates can be produced. Despite the need for automated log parsing that results from the exponential growth in size of log files, they also claim that privacy proves to be an obstacle in the path to automation. With log files capturing all activities within software systems, many companies, institutions, and organizations are hesitant to release their files for automation research, and most do not [4].

B. *"Drain: An Online Log Parsing Approach with Fixed Depth Tree"*

Also looking at log files are Pinjia He et al in their article, "Drain: An Online Log Parsing Approach with Fixed Depth Tree". One of their main reasons behind their work lies in their claim that most log parsing is done offline, which is time consuming. As such, they propose the use of online log parsing via Drain. Drain removes unnecessary time consumption by using a fixed depth tree. The performance of Drain supersedes other existing online parsers at the time, and, surprisingly too, offline parsers. The success rate of Drain roughly lies between 51% and 81%.

Some of their arguments mirror those in the aforementioned article, and, in fact, the two articles share common authors. Specifically, all four authors of this paper are also authors of the above research; however, the above paper has additional authors. Some similar claims include the utility of logs as a singular source of all runtime information of software systems, and the inefficiency of manually parsing logs that fall in the category of Big Data. In outlining this inefficiency, Pinjia He et al also cite the fact that global collaboration among developers occurs more frequently nowadays, which is an obstacle for manual parsing in which regular expression statements requires habitual verification.

The contribution of their paper encompasses the presentation of the design of Drain, the illustration of exhaustive real-world testing of Drain, and the public release of Drain's source code [7].

In general, most of the literature on log parsing points out that with the changes in time, the era of Big Data, automating log parsing methods is requisite. Consequently, a lot of investigation and experiments is taking place in the quest to achieve this goal.

III. Methodology

This section delves into the mechanisms that enabled the accomplishment of this work. These mechanisms range from initial data wrangling to Exploratory Data Analysis to Pre-Processing for Machine Learning.

A. *Preliminary Pre-Processing: Data Wrangling & Exploratory Data Analysis (EDA)*

To begin, the raw data was sourced from *ZENODO*, as per the directions on *Logpai/Loghub* [4][7]. The raw data came separated into two files, one of which contained the labels and that was well organized and structured. The other file, containing the features, was entirely in a raw an unstructured format. Consequently, an online scraper was sourced via *Logpai/Loghub* since this GitHub's project has as its primary goal the provision of an automated parser for log files. While the parser obtained, -Drain-, worked, the output was unavailable. Consequently, web scraping was done in search of another automated log parser. *IMB/Drain3* was found but proved tedious to implement given the time constraints of the project [2]. Subsequently, I wrote a parser based on the parameters of the raw log files and in conjunction with those in the "Demo" parser of *Logpai/Loghub* [5]. The parser I created passed the raw and unstructured HDFS log files into a DataFrame. From there, I began to investigate the state of the the data interms of "how clean" it was, and simultaneously carried out necessary cleaning. The DataFrame comprising of the labels was joined with the HDFS DataFrame. Notably, here, is that this DataFrame consisted of roughly half a million entries, which is a mere fraction of the over eleven million entries in the HDFS DataFrame. Further wrangling was carried out to properly match the labels with the entries in the HDFS DataFrame.

Thereafter, an Exploratory Data Analysis (EDA) was carried out to get some understanding of the features and their interactions with each other. One notable aspect of the data is its size, 11,175,629, which should prove unremarkable given that the goal here is to work with Big Data. Its significance lies in the fact that I would not be able to use my local computer when conducting Machine Learning techniques, which I learned only after trying and failing first. Thus, I employed *Google Colab* for these techniques [1].

B. *Pre-Processing and Pipeline Building (RandomForestClassification) for Machine Learning*

SciKit-Learn was employed to train our data to predict anomalous log files. At this stage, from the original features that came along with our data, "Component" and "Content" were dropped. Resulting from wrangling, seven features were used to predict the feature "Label" being abnormal or not. These covariates were immediately split into training and test set as a best practice step. Visualization of these covariates was then executed. A dummy classifier was used to fit the training dataset, and then cross-validation was carried out on the dummy model. The mean of the cross-validation was computed. The features were then separated according to their types - binary, categorical, ordinal and so on.

A pipeline was then made for each feature type, and a KNN-Imputer was used to make up for missing values. There was only one missing value in the data. Within the pipeline the data was also scaled using "median" as the strategy. A Random Forest Classifier was then used to build our model, with the goal of comparing its performance with the dummy classifier. This included making a pipeline to preprocess the data using a balanced approach, fitting, and cross validating the training

dataset. The mean scores of the results were computed for comparison. The hyperparameters were then tuned using RandomizedSearchCV for speed as opposed to GridSearchCV. Calculations were then done to find the best hyperparameter for maximum depth and the corresponding cross validation score. And a DataFrame was made using the cross-validation results from random_search. Random_search was also used to evaluate the model on both the full training and test sets. Using the test set I plotted a confusion matrix and printed a classification result of this set.

IV. EXPERIMENTS AND RESULTS

This section provides most of the results of the experiment at each stage of the experimentation and analysis.

A. Advanced Exploratory Data Analysis

The first five rows of the cleaned dataset from which I worked and from which I dropped our last columns is seen in Table I. In it are seven features: "Label", "block", "date", "time", "pid", "level", and "year". "Block" contains the unique identifying numbers. "Label" consists of binary categorical variables, namely normal and anomaly. "Pid" is inferred as referring to the process identification numbers since it was not explicitly stated in the literature. "Level" consists of binary categorical variables, namely info and warn. "Year" is an extracted subset of the "date" feature; both were kept for granularity.

Table II displays the head (first five rows) of our training set, which results from splitting the data to have an 80/20 split for the training and test sets respectively and a random state of 77.

Table III displays the distribution of the target values from the target feature, "Label". Deriving from the training data set, Table IV displays the separation of the feature vector from the target, "Label". It displays only the head of this result.

TABLE I. CLEANED DATA SET

Label	block	date	time	pid	level	year
Normal	1608999687919862906	2009-08-11	203518	143	INFO	2009
Normal	1608999687919862906	2009-08-11	203518	35	INFO	2009
Normal	1608999687919862906	2009-08-11	203519	143	INFO	2009
Normal	1608999687919862906	2009-08-11	203519	145	INFO	2009
Normal	1608999687919862906	2009-08-11	203519	145	INFO	2009

TABLE II. TRAINING SET

Label	block	date	time	pid	level	year
Normal	26971670885768519	2009-08-11	203904	32	INFO	2009
Normal	3689862003003839648	2009-08-11	203940	28	INFO	2009
Normal	8916560322184963168	2009-08-11	220401	13	INFO	2009
Normal	8054782785917830420	2009-08-11	214822	2785	INFO	2009
Normal	859705208416062320	2009-08-11	203603	30	INFO	2009

TABLE III. DISTRIBUTION OF TARGET VALUES

Label	
NORMAL	64078
ANOMALY	1726

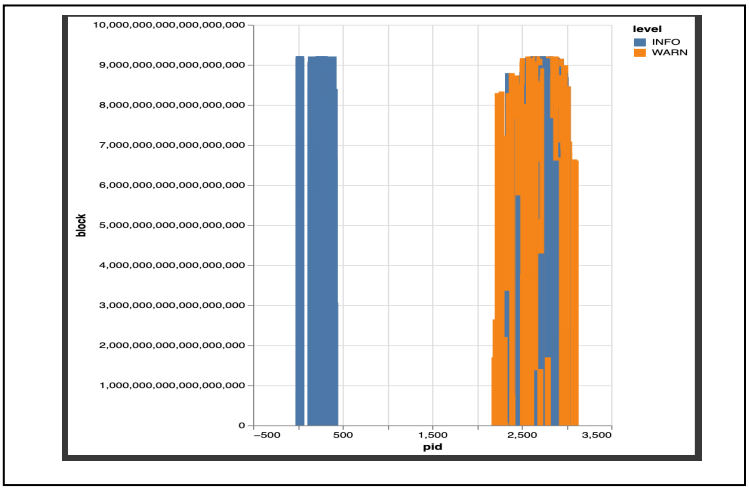
TABLE IV. SEPARATION OF FEATURES FROM TARGET

block	date	time	pid	level	year
26971670885768519	2010-08-11	203904	32	INFO	2010
3689862003003839648	2010-08-11	203940	28	INFO	2010
8916560322184963168	2009-08-11	220401	13	INFO	2009
8054782785917830420	2010-08-11	214822	2785	INFO	2010
859705208416062320	2009-08-11	203603	30	INFO	2009

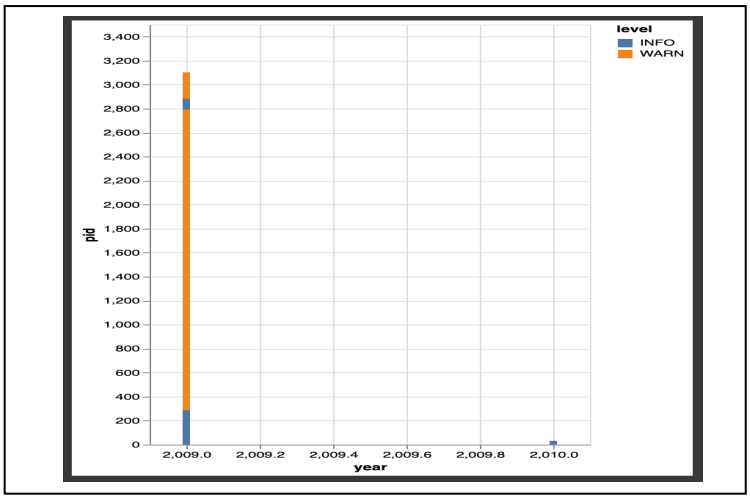
Plots I and II show two preliminary visualizations of the interaction among our covariate variables. Plot I depicts the interaction among the features of "block", "pid", and "level". It consists of two distinct colored regions; one is devoid of warn label and the other is almost filled with it. With respect to the identifying numbers of the feature "block", warn levels are present throughout.

Plot II illustrates the interaction among three features, "pid", "level", and "year". I note here that our complete dataset spans three years, 2009 to 2011. From this plot, it appears that only the years 2009 and 2010 consist of the feature "level". The occurrence of warn levels in 2009 far outnumbers that of info levels by approximately eighty percent. This decreases in 2010 and, in fact, very little representation is seen in the plot for the feature "level".

PLOT I.



PLOT II.



B. *Model Building Results*

The results of the cross-validated scores from the Dummy Classifier, predicated upon a strategy of "most frequent", are in Table V. It contains the time it took the dummy model to fit onto the training set, the scores for the time, and the scores for the training and test sets. Table VI consists of the mean of these scores.

The results of the cross-validated scores from Random Forest Classification model, predicated upon a strategy of "balanced" weights, are in Table VII. It contains the time it took the Random Forest Classification model - RFC model- to fit onto the training set, the scores for the time, and the scores for the training and test sets. Table VIII consists of the mean of these scores.

The hyperparameters of Random Forest Classification model were tuned and the optimal parameters and optimal score were computed. Table IX illustrate these values.

TABLE V. DUMMY MODEL CV SCORES

fit_time	**score_time**	**test_score**	**train_score**
0.037651	0.028269	0.973786	0.973767
0.039370	0.028632	0.973786	0.973767
0.035837	0.029904	0.973786	0.973767
0.036243	0.028841	0.973710	0.973786
0.035870	0.027950	0.973784	0.973767

TABLE VI. MEAN OF DUMMY MODEL SCORES

fit_time	0.036994
score_time	0.028719
test_score	0.973771
train_score	0.973771

TABLE VII. RFC MODEL RESULTS

fit_time	**score_time**	**test_score**	**train_score**
0.803225	0.056542	0.990350	0.999430
0.832269	0.057933	0.989135	0.999278
0.788882	0.054822	0.988907	0.999297
0.822543	0.053885	0.988755	0.999335
0.803076	0.053754	0.989514	0.999582

TABLE VIII - MEAN OF RFC CV SCORES

fit_time	0.809999
score_time	0.055387
test_score	0.989332
train_score	0.999385

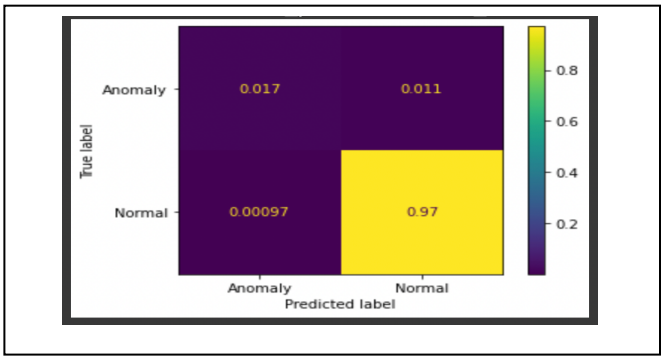
TABLE IX. RFC: OPTIMAL PARAMETERS & OPTIMAL SCORE

randomforestclassifier__max_depth	19
randomforestclassifier__n_estimators	278
Optimal Score	0.9868701073899527

TABLE X: CLASSIFICATION REPORT - TEST SET

	Precision	Recall	F1-Score	Support
Anomaly	0.95	0.60	0.74	474
Normal	0.99	1	0.99	15977
Accuracy			0.99	16451
Macro Avg	0.97	0.80	0.86	16451
Weighted Avg	0.99	0.99	0.99	16451

PLOT III - CONFUSION MATRIX



C. Evaluation of Random Forest Classification Model

This model was evaluated on the full training and test sets. A confusion matrix plot was created based on the test set and is seen in Plot III. A classification report on the test set was also generated and is seen in Table X.

REFERENCES

[1] *Colab*. *Google*. < https://colab.research.google.com >.

[2] *IBM/Drain3*. < https://github.com/IBM/Drain3 >.

[3] *Hadoop*.<https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html#Introduction>.

[4] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, Michael R. Lyu. "Tools and Benchmarks for Automated Log Parsing". *ICSE International Conference on Software Engineering*, 2019.

[5] "Loghub". *LOGPAI*. <https://github.com/logpai/loghub>.

[6] Pinjia He, Jieming Zhu, Shilin He, Jian Li, Michael R. Lyu. "An Evaluation Study on Log Parsing and Its Use in Log Mining". *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016.

[7] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. " Drain: An Online Log Parsing Approach with Fixed Depth Tree". IEEE 24th *International Conference on Web Services*, 2017.

[8] *ZENODO*. < https://zenodo.org/record/3227177#.Ym3Ob5LMJQI>