Anomaly Detection Using Logistic Regression

Authors: Connor Fairbanks, Ben Gao
Date: May 8, 2021

Abstract

Anomaly detection in log files can be a challenging and time consuming task. With the use of basic data wrangling techniques (regular expressions), and a logistic regression binary classifier, a model can be trained on log data to detect anomaly log events with high accuracy. Although many different approaches can be employed, this paper describes a simple and effective solution that uses the python programming language (and associated libraries) to save time in the task of anomaly detection.

Introduction

A log file is defined as “a computer-generated data file that contains information about usage patterns, activities, and operations within an operating system, application, server or another device” [1]. These log files can provide those individuals in charge of maintaining a system with valuable information regarding the state of that system. A problem begins to arise as these log files begin to grow in size. Depending on the system, there can be hundreds of thousands of log files created each day. The task of filtering through these files and gleaning useful information from them quickly becomes unmanageable for a human without the help of machine learning.

The end goal of the analysis of log files is to pinpoint the times/events where the system is not running properly. If one can find these events, then they can gather more information about the event and attempt to diagnose the problem and correct it. When there are millions of lines of text to search through, finding the anomalies can be a time consuming task. If anomalies are rare, the task of locating them can be compared to finding a “needle in a haystack” [2]. Another complicating factor is the fact that not all log files have a uniform structure, so there is not a uniform method for analyzing them.

In this paper we will discuss one approach for anomaly detection in the HDFS log files. The raw log files used in this paper were found on Zenodo.org [3]. HDFS stands for Hadoop Distributed File System and it “is the primary data storage system used by Hadoop applications”[4]. Hadoop “is an open source distributed processing framework that manages data processing and storage for big data applications”[4]. The log files for big data applications like these can contain millions of lines of text. With the use of data cleaning techniques found in the pandas library of the Python programming language, this large amount of data can be organized into a format that can be fed into a machine learning model that will be able to classify the blocks found in the log file events as either ‘normal’, or as an ‘anomaly’. The classification technique known as logistic regression was used in order to classify the blocks in each event. Logistic regression is a supervised learning technique that is well-suited for binary classification tasks like the one described here. Another reason that the authors chose this approach was that they were more familiar with the concepts associated with logistic regression as opposed to other classification techniques/deep learning approaches.

Literature Review:

Our cleaned data contains a number of categorical features that needed to be encoded in order for a machine learning algorithm to successfully utilize them. One popular method for encoding such data is called one-hot encoding. One-hot encoding takes categorical variables and encodes them into a suitable feature vector [5]. For example, if a variable had the categories of {*blue, green, red}*, one-hot encoding would encode them respectively as {[1,0,0], [0,1,0], [0,0,1]}. This results in a vector space where each category is “orthogonal and equidistant to the others” [5]. This type of encoding avoids the problems associated with ordinal encoding. If the colors were encoded 1, 2, and 3, then it implies that there is a type of ordinal relationship between the three colors, when really there is not (they are nominal categories). This improper relationship can have a negative effect on the modelling [6]. One downside of one-hot encoding is the increase in the dimensionality of the data [5], but that did not seem to be an issue in our case.

Once the data has been encoded, it can be used in a machine learning algorithm. Due to the binary nature of this project (“normal” or “anomaly”), we decided to use a logistic regression classifier to classify our events. Logistic regression is a useful method because it yields a probability that a given observation will belong to either the “normal” or “anomaly” category [7]. We can use this probability to set a threshold at 0.5, and any value with a probability higher than 0.5 is classified as “normal”, and any value with a probability less than 0.5 is classified as “anomaly”. The behavior of the logistic regression can be seen below in figure 1 [7].

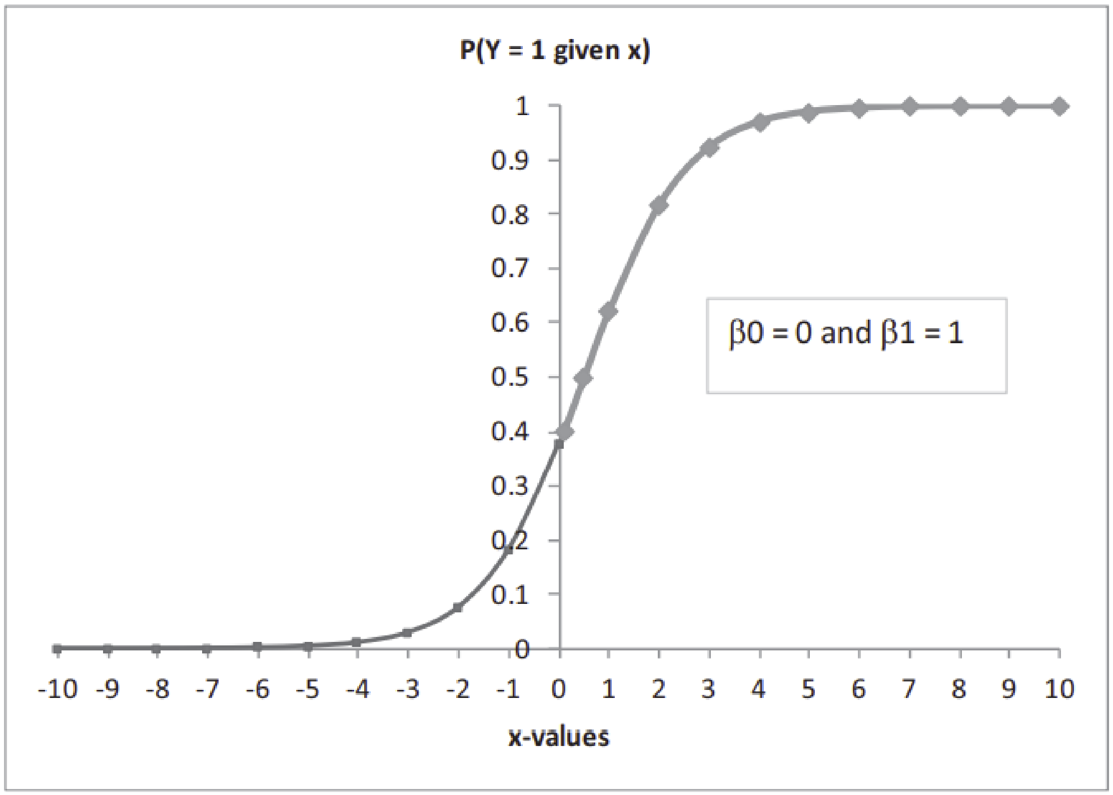


Figure 1: Behaviour of Logistic Regression function.

In an unbalanced data set such as ours (with few “anomaly” blocks compared to “normal” blocks), the simple misclassification rate can be a misleading measure of model performance. For example if out of 100 events, only 2 are categorized as an “anomaly”, then the model can simply categorize all events as “normal” and attain a very low misclassification rate of only 2%. Rather than using the misclassification rate as our measure of accuracy, we can use the F1 score. The F1 score is defined as:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The F1 score has been shown to be a more reliable measure of model performance than misclassification rate for unbalanced data sets [8].

Background

Many different methods can be used for anomaly detection in the past. He et. al [9] discussed several of these methods including supervised (Logistic Regression, Decision Tree, and Support Vector Machine) and unsupervised techniques (Log Clustering, Principal Component Analysis, Invariants Mining). He et al [9] also showed that out of the supervised techniques, that Logistic Regression significantly outperformed the other methods in terms of F1 score (depending on the window technique).

Studiawan et. al also showed that Gated Recurrent Unit (GRU) networks could be successfully used to perform anomaly detection on log files with an F1 score of 99.84% [10]. In this paper, due to the experience and knowledge level of the authors, we decided to approach this problem with a Logistic Regression classification approach.

Methodology

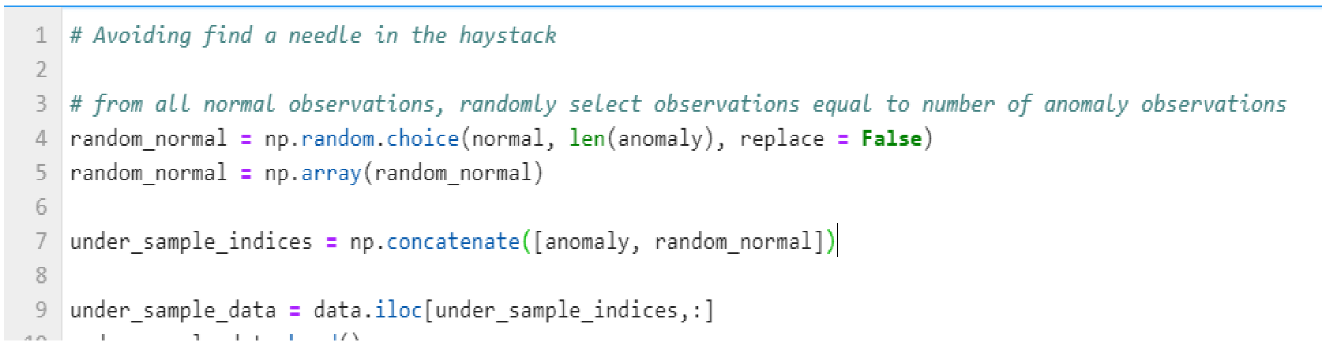
The initial step in the process of building this machine learning pipeline was the exploratory data analysis. By uploading the log file into a jupyter notebook, we were able to use the python programming language to organize the data. The log file was in the format of a text file, with one event logged per line of text. There were 11175629 lines of text (events) in the file. By visual inspection of the lines of text, we began to notice common themes that were present in all of the events (also known as an event schema). All events contained the following pattern:

- A 6 digit number (e.g. 081110) which could have been a date stamp for the day the event was created.
- A second 6 digit number. This was possibly another portion of the timestamp of the event.
- A third number (of varying lengths of digits)
- A binary message (INFO or WARN).
- A string starting with “dfs.” and ending with a “:” (there were only 9 different types of these in the entire log file).
- A single unique block was mentioned (sometimes the same block was mentioned twice).
- A string message about the event that varied based on the type of event it was (the “dfs…:” type).

The technique using regular expressions was used to determine that these aspects were present in all lines of the log file. We also used word count “for” loops in order to determine how often certain words were showing up in each type of event. Regular expressions were also used to determine that there were 9 different types of events (based on the “dfs…:” string). Based on the techniques described by Larry Lancaster [11], we decided to bring structure to the data by creating 9 separate dataframes (one for each of the event types). We then used regular expressions again to find the attributes that were unique to these 9 separate dataframes. We organized each data frame so that the non-numerical columns were categorical, and each row fell into one of the categories listed. The end goal was to merge the 9 dataframes into one final dataframe that had each row as one event in the log file, with the unique block id mentioned in that row as the index of the row of the dataframe. The rest of the columns would then contain information about the event, and the categorical attributes could be one hot encoded to a numerical format that could be understood by the logistic regression algorithm.

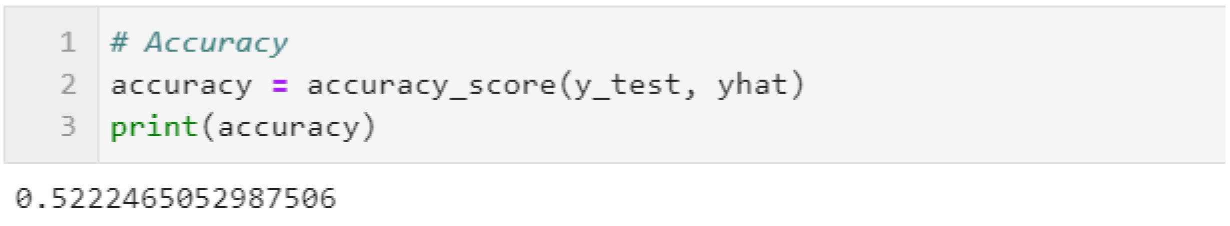
This source of the log files also provided a file that labelled each individual block as either an “anomaly” block, or a “normal” block. After merging this labelling to the large dataframe of all of the log events (with the unique block ID as each row index), we discovered that there was only about 2.6% of events that were anomalies. This created a problem for us because when the logistic regression model was trained on data that was this unbalanced, it determined that all events should be classified as “normal” in order to gain a high accuracy rate. This approach resulted in a high number of false negatives (“anomaly” blocks labelled as “normal” blocks). A simple approach was used to overcome this problem of an unbalanced data set.

In order to avoid having our model look for and fit 282927 anomaly observations from a total of 11170084 observations, we used needle and haystack approach [12]. The implementation is first filtering out and counting all the anomaly observations, then randomly sampling out an equal number of observations from the normal observations. Therefore, our resulting data subset should have an equal number of normal observations and anomaly observations.


```python
# Avoiding find a needle in the haystack

# from all normal observations, randomly select observations equal to number of anomaly observations
random_normal = np.random.choice(normal, len(anomaly), replace = False)
random_normal = np.array(random_normal)

under_sample_indices = np.concatenate([anomaly, random_normal])

under_sample_data = data.iloc[under_sample_indices,:]
```

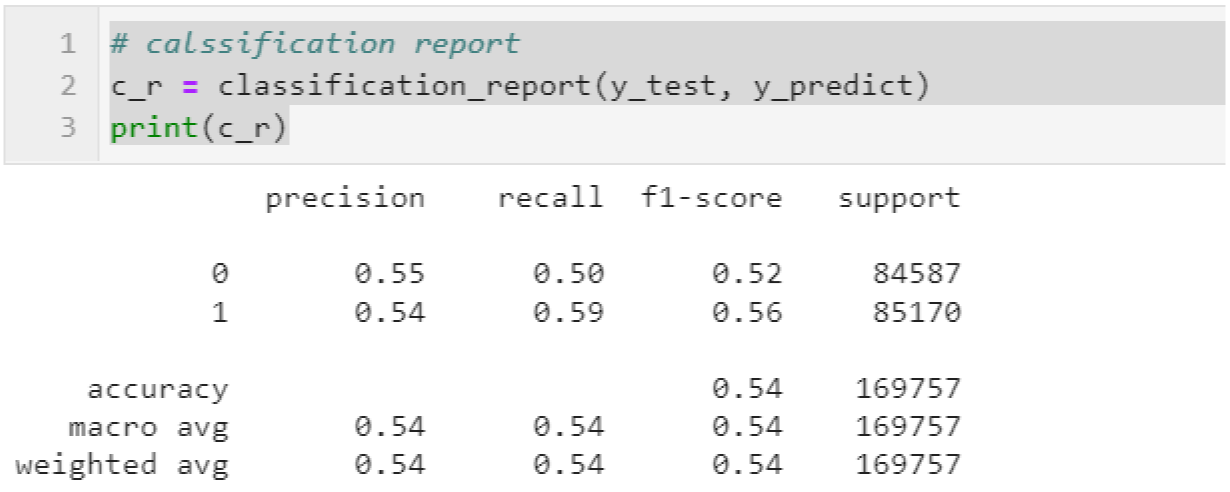
Experimentation and Results

Before we apply the needle and haystack approach, the performance of our model is only slightly better than randomly guessing. The accuracy is only 0.52,
And the confusion matrix suggests that the observations are categorized evenly between normal and anomaly by our model.



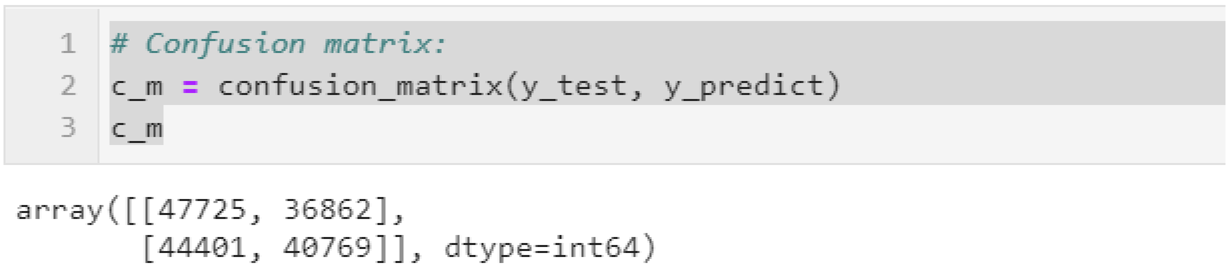
```
# Accuracy
accuracy = accuracy_score(y_test, yhat)
print(accuracy)
```

```
0.5222465052987506
```



```
# calssification report
c_r = classification_report(y_test, y_predict)
print(c_r)
```

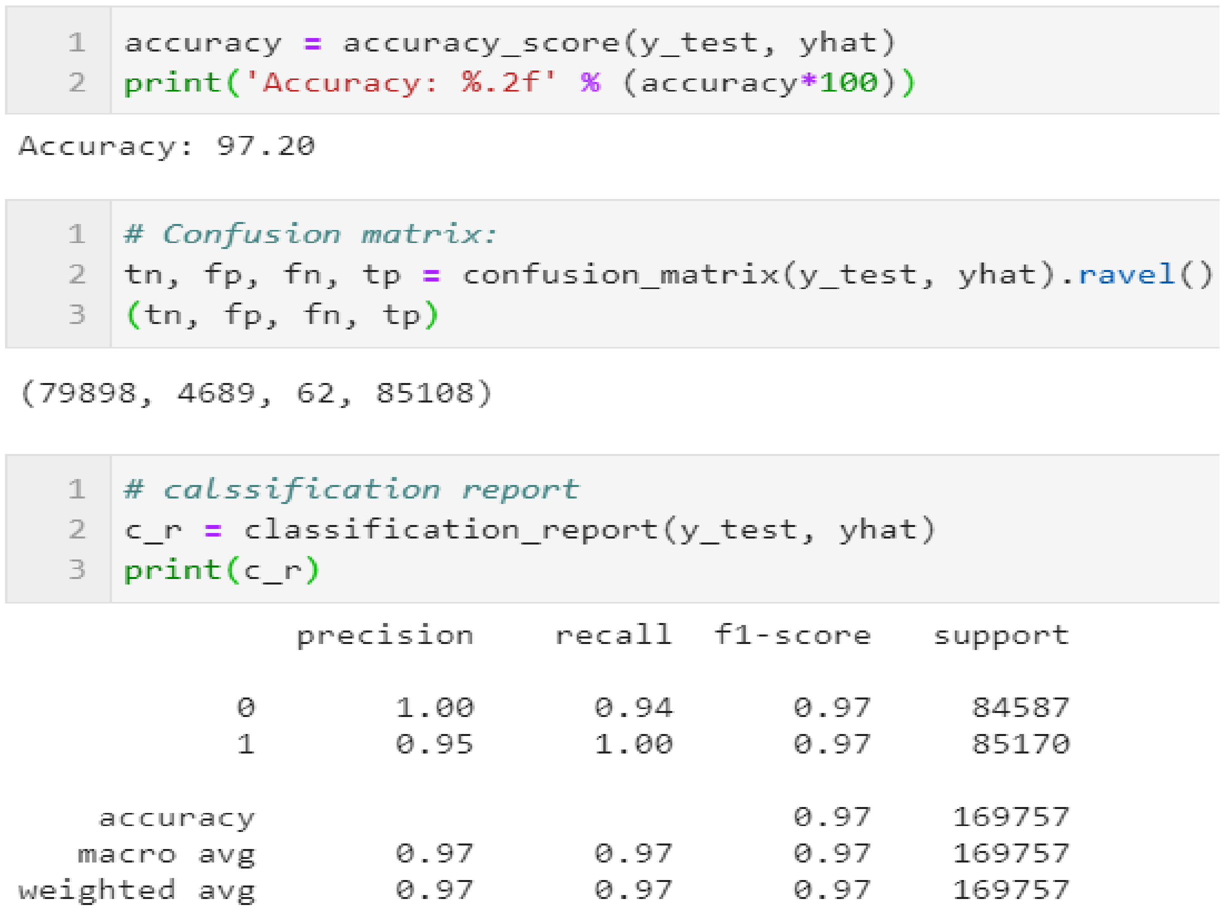
```
              precision    recall  f1-score   support

           0       0.55      0.50      0.52     84587
           1       0.54      0.59      0.56     85170

    accuracy                           0.54    169757
   macro avg       0.54      0.54      0.54    169757
weighted avg       0.54      0.54      0.54    169757
```



```
# Confusion matrix:
c_m = confusion_matrix(y_test, y_predict)
c_m
```

```
array([[47725, 36862],
       [44401, 40769]], dtype=int64)
```

After applying the needle in haystack approach, the subset of our data is much more balanced. As a result, our model performance is significantly improved. The accuracy comes up to 97% and the false positive and false negative value decreases sharply. This result implies that our needle and haystack approach works very well.



```python
accuracy = accuracy_score(y_test, yhat)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 97.20
```

```python
# Confusion matrix:
tn, fp, fn, tp = confusion_matrix(y_test, yhat).ravel()
(tn, fp, fn, tp)
```

```
(79898, 4689, 62, 85108)
```

```python
# calssification report
c_r = classification_report(y_test, yhat)
print(c_r)
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97     84587
           1       0.95      1.00      0.97     85170

    accuracy                           0.97    169757
   macro avg       0.97      0.97      0.97    169757
weighted avg       0.97      0.97      0.97    169757
```

Discussion

The technique we employed yielded acceptable results. We were able to overcome the problem of an unbalanced data set by selectively choosing a similar amount of data from events labelled as “anomaly” and events labelled “normal”. This may have worked in this scenario because we had so much data, but our model could have benefitted by taking a different approach so we could use all of the available data to train the model. This may allow our model to perform better on future data.

As we have learned more about this problem through experience and research, we have found there are many areas where our approach and techniques could be improved. During the data wrangling stage, we merged the events data frame (with block ids mentioned in each event as the row index) with the block label data frame. These two data frames had drastically different numbers of rows, so the result was a large data frame with a single block id showing up in multiple rows. If we could approach this project again, we would take a similar approach as mentioned by He et al. [9] and have each unique block be its own vector, and the type of event that those blocks appeared in would make up the vector. For instance if we had a row in the data frame for block A, the columns would be the 9 different types of events (“dfs…:”) and the message (“INFO”/”WARN”), and any other columns we want. The number of times that block was involved in events that contained those features, that would be the number in each cell. We could then normalize those cell values and we would end up with a dataframe with one row per block. This would allow us to merge the events data with the block labels data and have the same number of rows.

Another approach that we would change in the future would be to use a log parsing library and a deep learning approach. This may allow us to build a product that could be used on many different types of log files. By using regular expressions, this solution is highly tailored to the HDFS log files that we used in this paper, and altering the regular expressions to work with a different type of log file (with a different schema), would take a considerable amount of time and effort.

Conclusion

We have shown that through the use of regular expressions, data wrangling, as well as binary classification using logistic regression, that anomaly detection can be successfully performed on log files. The initial goal was achieved and a model was created that can successfully classify log events as “normal” or “anomaly”. The approach taken in this paper yields an adequate result, but it would require significant alteration in order to be used on a wider variety of log files with different schemas.

References

1. Sumo Logic, Inc. (2020, April 29). *What is a Log File?* Sumo Logic. https://www.sumologic.com/glossary/log-file/

2. Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. (2009). Detecting large-scale system problems by mining console logs. Paper presented at the 117-132. https://doi.org/10.1145/1629575.1629587

3. Team, H. B. L. (2018, January 1). Loghub. Zenodo. https://zenodo.org/record/3227177#.YJM6AsCSmUk

4. Rosencrance, L., Vaughan, J., & Preslar, E. (2021, March 2). *Hadoop Distributed File System (HDFS)*. SearchDataManagement. https://searchdatamanagement.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS

5. Cerda, P., Varoquaux, G. & Kégl, B. Similarity encoding for learning with dirty categorical variables. *Mach Learn* 107, 1477–1494 (2018). https://doi-org.ezproxy.library.ubc.ca/10.1007/s10994-018-5724-2

6. Brownlee, J. (2020, August 17). Ordinal and One-Hot Encodings for Categorical Data. Machine Learning Mastery. https://machinelearningmastery.com/one-hot-encoding-for-categorical-data/

7. Logistic regression for binary classification. (2019). In K. V. S. Sarma, & R. V. Vardhan (Eds.), (1st ed., pp. 169-184). CRC Press. https://doi.org/10.1201/9780429465185-9

8. Andrews, J. (2021, January 11). *DATA 572 Supervised Learning: Performance Indices* [Slides]. Canvas. https://canvas.ubc.ca/courses/64332/files/12266297?module_item_id=2818433

9. He, S., Zhu, J., He, P., & Lyu, M. R. (2016). Experience report: System log analysis for anomaly detection. Paper presented at the 207-218. https://doi.org/10.1109/ISSRE.2016.21

10. H. Studiawan, F. Sohel and C. Payne, "Anomaly Detection in Operating System Logs with Deep Learning-based Sentiment Analysis," in IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/TDSC.2020.3037903.

11. Lancaster, L. (2020, August 5). *Machine Learning for Anomaly Detection in Log Files | Zebrium*. Zebrium. https://www.zebrium.com/blog/using-machine-learning-to-detect-anomalies-in-logs

12. Alam, M. (2020, November 1). *Supervised Machine Learning Technique for Anomaly Detection: Logistic Regression*. Medium. https://towardsdatascience.com/supervised-machine-learning-technique-for-anomaly-detection-logistic-regression-97fc7a9cacd4.