Anomaly Detection on HDFS Log Files

ABSTRACT

Anomaly detection is an important aspect of managing large scale file systems that are sourced from distributed systems. Log monitoring is one of the key areas of research and evaluating them as they are being recorded gives us actionable insights. Log messages are now broadly used in cloud and software systems. With the increased scale and complexity of the data coming in as logs, manual inspection of them seems infeasible. There are several detection methods based on automated log analysis. However, the area of anomaly detection in the context of machine learning models seems relatively untouched. Our approach aims to perform feature extraction from the templates obtained from the logs and deploy machine learning models for anomaly detection. We also provide a summary of the performance of the models used. This paper gives a detailed approach of how we use an unstructured HDFS log file through a LogParser library that uses an unsupervised learning algorithm to generate structured log files as an output. The event templates are labelled based on BlockIds that make up a trace. We then utilize PCA in order to achieve dimensionality reduction as well as address multicollinearity issues. We then implement Logistic Regression, KNN, Quadratic Discriminant Analysis, Random Forests, and Gradient Boosting Decision Tree classification models using a rigorous evaluation framework with weighted binary cross entropy loss and F1 score metrics to account for severe class imbalance. We find that the tuned gradient boosted classification tree achieves the highest CV F1 score and lowest CV cross entropy loss. Finally, we evaluate this model using a withheld 15% test set in order to estimate the expected real-world performance, achieving a 0.9978 test F1 score and 0.0049 test cross entropy loss.

1. INTRODUCTION

Logs are widely used to record machine runtime information, such as timestamps, ID, levels and log components, and the current state of the log entry. Gaining visibility into modern IT and software environments is a challenge that a number of organizations are finding difficult to overcome. In today's growing digital world, almost all systems leave a digital footprint of their operational status, configurations, environmental changes and errors into an event log of a particular template. This gives us an excellent chance to understand the operational status of the systems, health and performance of the internal programs, computing infrastructure, network and security analysis of downtime incidents.

Data centers consist of thousands of software components that report a million misbehaves. Traditionally developers use print commands and complex monitoring libraries to record trace execution and runtime statistics. Large scale services use multiple log parser libraries[1] that produce structured information from the logs coming from various sources. Typically, a log message records a specific system event with a set of fields: *timestamp(*recording the occurrence of the event), severity level, message content, template and components. The raw message can be split into two parts: constant part and the variable part. This is classified as the event template for different log events in the dataset. For instance, The sentence " I am going to the mall" will be parsed into the template " I am going to the <*>" with "mall" as the parameter. The constant part constitutes the fixed plain text and remains the same for every event occurrence, which can reveal the event type of the log message. The variable part carries the runtime information of interest, such as the values of states and parameters (e.g., the IP address and port: 10.251.31.5:50010), which may vary among different event occurrences. The goal of log parsing is to extract the event by automatically separating the constant part and variable part of a raw log message, and further transform each log message into a specific event. Log parsing heavily relies on regular expressions.

2. BACKGROUND AND LITERATURE REVIEW

Anomaly detection is the task of finding observations that are not conformed as the normal or the expected behaviour. In simple terms, these observations are named as outliers, novelty, surprises and anomalies in different applications. With the advent of large scale computing systems, anomalies are a problem in various areas such as Intrusion detection, Industrial damage detection, Medical and public health, Anomaly detection in text data and sensor networks. Anomalies can be broadly classified into three categories - Point anomalies, contextual anomalies and collective anomalies. Traditional approaches that developers use (keyword searches - "fail",'exception") or regular expressions match fail in large scale distributed systems. This kind of anomaly detection relies heavily on manual inspection of logs that have become inadequate. The reasons are due the following -

1) Large scale modern systems behave too complex to be comprehended by developers. Thousands of lines of data are sourced and live streamed using parallel computing systems (eg.Spark and Hadoop)
2) Modern systems generate logs at the rate of 50 gigabytes per hour[8]. The volume of incoming logs turns out to be incomprehensible for manual discretion.
3) With the advent of systems, fault tolerance mechanisms are employed. This contributes to redundant entries and even proactively kills a speculative task to

improve performance. This significantly leads to increased efforts in manual inspection.

Due to the above reasons, automated log analysis methods for anomaly detection are in high demand. There are many automated methods that are designed and developed, however there seems to be a gap in understanding the nature of logs and implementing machine learning models for predictions. Some of the approaches that can be useful in finding anomalies can be classified into three broad categories - Supervised, semi-supervised and Unsupervised anomaly detection. This paper takes a supervised approach to anomaly detection on the log file used in the logpai package(source). HDFS is the Hadoop Distributed File System designed to run commodity hardware that generates system logs. The HDFS_1 is a log set that is generated in a private cloud environment using benchmark workloads and labeled manually using another file. Machine learning models are deployed on a parsed log file and metrics are evaluated on important features. A comparison is taken for the models and a best model based on precision, recall, accuracy and weighted cross entropy. We used the log parser library to output a structured log file, processed with feature extraction to arrive at an event count matrix with 156 unique block entries. Each line is considered an event log. The data point and its individual words are considered attributes, then the job of clustering similar log reduces to grouping similar logs together[3].The matrix is then standardized and tested for multicollinearity using PCA. This throws a PCA score matrix explaining the maximum percentage of the variation in the first 90 components. The results from logistic regression, QDA, K- Nearest Neighbours, Random Forests and Gradient Boosting Classifier are provided in a comparison table.

Some of the key research that was taken for reference relates to the work on unsupervised learning on data mining and mining. Some of the automated log mining that was researched and understood were SLCT[2], IPLoM [3], LKE[4], LogSig[5]. With discretion on relating the current study of our HDFS logs, we went with the IPLoM as the best choice for the log parsing[1]. A detailed understanding of the IPLoM and the methodologies was taken from the clustering analysis and data mining. Several algorithms like CLIQUE, CURE and MAFIA have been designed for clustering high dimension data[2][16]. The IPLoM works on a 2-Step hierarchical partitioning process that produces cluster descriptions or line formats for each of the clusters. The results using IPLoM are promising with results on recall and precision of 0.81 and 0.73 respectively. A detailed review of the baselines associated with anomaly detection was studied with log parsing, log collection, feature extraction and anomaly detection[7]. Understanding how different logs produce anomalies is crucial in deciding the models we intend to use for its prediction. Their performance also depends on the kind of

software systems we use.[17]. Some popular model choices after parsing the file are Logistic Regression, Decision Trees, and SVMs.[7]

3. METHODOLOGY

In this section, we will describe the detailed process of the steps we took to process the HDFS log files, arrive at our modelling choices, and outline how we will tune and evaluate our models.

3.1. Log Parsing: LogParser

Some of the automated log parsing methods include SLCT[2], IPLoM [3], LKE[4], LogSig[5] to name a few. For our analysis, we used the IPLoM(Iterative Partitioning Log Mining) algorithm. This algorithm uses unsupervised learning for mining of clusters from event logs. [6] IPLoM uses a three step hierarchical process that partitions log data into its respective clusters. In the final step IPLoM produces line formats for each of the events. It is designed as a log data-clustering algorithm. It works by iteratively partitioning a set of messages used for training. At each step of the partitioning process the resultant partitions come closer to containing only log messages, which are produced by the same line format. At the end of the partitioning process the algorithm attempts to discover the line formats that pro- duced the lines in each partition, these discovered partitions and line formats are the output of the algorithm. The four steps of which IPLoM goes through are:

1. Partition by token count.
2. Partition by token position.
3. Partition by search for bijection.
4. Discover cluster descriptions/line formats.

The dataset used in this paper is sourced from here. (LINK). The algorithm works in the same way, where logs are labelled based on BlockId and each collection of entries makes up a “trace”. The parsed log file provides the event templates that contain the regex of the raw messages. The columns contained in parsed file are :

1. LineId - line entry of the event
2. Date - Date of the event recorded
3. Time - exact time of occurrence of the event
4. Level - verbosity level/ severity of the model
5. Pid - product Id
6. Component - the raw message component

7. Content - messages
8. EventId - unique id corresponding to the event occurrence
9. EventTemplate - The format depicting the framework of the incoming messages
10. ParamterList - important list of indices

This parsed file is merged with the anomaly labels that are provided on LogHub(LINK). This file links the EventId in the parsed file with the block Ids that contribute to a trace. Each trace contains 20 entries on an average with a range of 2 - 298. This gives us about ~575000 unique block Ids that are associated with anomaly labels. This combined fine is taken as our dataset for further analysis.

3.2. Feature Extraction

For feature extraction, we use the loglizer library. Loglizer provides a toolkit for implementing various machine-learning based log analysis techniques for automated anomaly detection. [7] The framework consists of the log collection, log parsing, feature extraction and Anomaly detection. The “FeatureExtractor" gives an event count matrix of n*p dimensions with no normalization. The matrix shape is synonymous to the total number of event entries and features considered.

3.3 Data Preprocessing

The HDFS log files contain a total of 156 unique event types. After performing feature extraction, we are left with a feature matrix of size 575061x156. Given the high dimensionality and relatively sparse nature of this dataset, we decided to perform dimensionality reduction in order to make our dataset more conducive to modelling and reduce the computational complexity of our model fits. We also wanted to address the strong multicollinearity in our dataset, as this poses assumption violation issues for models such as logistic regression and quadratic discriminant analysis that we intended to implement. Given these goals, we decided to apply Principal Component Analysis (PCA) after standardizing our data to have a mean of 0 and unit variance (this ensures that high variance features are not given unjustifiably high weight in the resulting components). Principle components are linear combinations of all of the original predictors, allowing us to achieve our goal of dimensionality reduction of our feature matrix while still retaining all of the original variables. We chose to retain enough principle components to account for 99% of the variation in the training set, which resulted in the selection of the first 90 principle components, a significant decrease in dimensionality in comparison to our original 156. Beyond this, the resulting scores from PCA are not correlated with each other, thereby eliminating the multicollinearity issue. Component loadings were computed using only our training dataset and then applied to

produce scores for both our training and test sets, thereby ensuring that information was not leaked from our test set in the process (as was the case for our standardization).

3.4 Model Evaluation Framework

It was crucial for us to consider the highly imbalanced nature of our training dataset in our choice of evaluation metrics. Given that appx. 2.93% of the samples are anomalies, we sought to choose metrics that would take into account this imbalance and ensure that our models were not overpredicting the dominant class. Measures such as accuracy could paint a misleading picture, given that a model that always predicts no amamoly would attain an accuracy of appx. 97.07% With this in mind, we selected evaluation metrics which are robust to this class imbalance: **weighted binary cross entropy loss** (used for hyperparameter tuning) and **F1 score** (used for model selection). **Recall**, **precision**, and **accuracy** are also reported. The formula for the weighted binary cross entropy loss can be written as follows:

$$Error = -\ [w\ *\ (P_{true})(log(\hat{P_{true}}))\ +\ (1\ -\ P_{true})(log(1\ -\ \hat{P_{true}}))]$$

Here, we set the weight $w$such that the loss associated with incorrect predictions when $P_{true} = 1$(the true class is “amamoly”) is weighted equal to $\frac{\#\ total\ samples}{\#\ anomalies}$.This has the practical effect of mimicking a balanced dataset by treating each positive sample as worth approximately 34 negative samples. Cross entropy loss has the added benefit of taking into account a model’s certainty: unlike metrics such as accuracy or recall which consider only the binary prediction, this formula takes the log of the probability assigned to the correct class, causing loss to decrease as the probability assigned to the correct class increases. This means that, for example, assigning a probability of 90% to the correct class results in a lower loss than a probability of 60%, thereby promoting certainty in addition to correctness. The F1 score takes the harmonic mean of the precision and recall, allowing us to ensure that our model both identifies a high proportion of anomalies as well as attains a high degree of accuracy for cases in which the true class is an anomaly. This metric is robust to class imbalance by excluding performance on true negatives.

We implemented a robust evaluation framework in order to allow for rigorous testing of our models. We conducted an 85/15 train/test split, with the test set withheld throughout our model selection and tuning process in order to attain an accurate assessment of our final model’s expected real-world performance. Hyperparameter tuning was conducted using a combination of random and exhaustive grid search (depending on the model fit time as well as the size of the search space) with 5-fold cross validation. Parameters were selected to minimize the average weighted binary cross entropy loss across all 5

folds. Model selection was conducted primarily using the cross-validated estimate of the F1 score.

3.5 Modelling Approach

In order to develop our anomaly detection model, we implemented five approaches: **Logistic Regression**, **KNN**, **Quadratic Discriminant Analysis**, **Random Forests**, and **Gradient Boosting Decision Trees**. Our selection of these particular models was motivated by a desire to examine models of varying degrees of complexity, which span a wide range of the inference-predictive power tradeoff. Our overall philosophy was to prioritize predictive power over explainability, given that the primary goal of our model is the real-time detection of anomalies rather than the drawing of inferential conclusions surrounding what log events are most likely to be associated with an anomaly.

3.6 Tools

As previously outlined, we used the LogParser library in order to parse the raw HDFS log files, as well as the Loglizer library in order to perform feature extraction using the parsed logs and generate an event count matrix. All of our modelling and model evaluation was done using the Scikit-learn package in Python. Random seeds were used whenever possible to ensure reproducibility of our results.

4. EXPERIMENT AND RESULTS

Model	CV F1 Score	CV Binary Cross Entropy Loss	CV Accuracy	CV Recall	CV Precision	Avg Fit Time	Avg Score Time
Tuned Gradient Boosting	0.998014	0.016280	0.999880	0.999160	0.996870	598.124890	0.298890
Gradient Boosting	0.997839	0.018840	0.999870	0.999020	0.996660	313.383830	0.287480
Random Forest	0.997804	0.018870	0.999870	0.999020	0.996590	58.082470	0.324100
Tuned Random Forest	0.997803	0.017690	0.999870	0.999090	0.996520	48.847250	0.490960
Tuned KNN	0.997599	0.027390	0.999860	0.998540	0.996660	0.186730	2072.159790
KNN	0.993185	0.123140	0.999590	0.993180	0.993190	0.017140	24.273790
Logistic Regression	0.989705	0.110330	0.999390	0.994140	0.985310	37.739170	0.075110
QDA	0.942036	0.903340	0.996530	0.950480	0.933740	18.955640	0.354000

4.1 Model Outlines and Results

4.1.1 Logistic Regression

For our first model, we implemented binary logistic regression [10][11]. This uses a generalized linear model to model a linear relationship between the log odds $log(\frac{p}{1-p})$

and the predictors. Unlike simple linear regression, this has the practical effect of ensuring that our model output is a value between 0 and 1, thereby allowing us to interpret the output as the probability of a particular trace being an anomaly (a threshold of 0.5 is used to produce binary output labels). This makes logistic regression a relatively simple model, with high explainability, low computational complexity to fit, and relatively low variance (not prone to overfitting). Though logistic regression does not make many key assumptions of general linear regression models, there are some assumptions that still apply. A binary logistic regression requires that the dependent variable or the predictors hold a binary value. An ordinal binary regression requires the dependent variable to be an ordinal one. The second assumption for logistic regression is that the observations should not come from repeated measurements or matched data. The third assumption requires that there be very little multicollinearity among independent variables. This is one of the key reasons for us to perform the test of multicollinearity using PCA and cross entropy functions. The fourth and the key assumption is that logistic regression assumes linearity of independent variables and log odds. It requires that the independent variables are linearly related to the log odds.

5-fold cross validation was used to evaluate the performance of this model, as outlined in section 3.4. This model achieved a weighted binary cross entropy loss of 0.110, with an F1 score of 0.9897. The very high CV performance of this relatively simple model suggests that our event count features are able to easily explain whether or not a particular trace contains an anomaly.

4.1.2 Quadratic Discriminant Analysis

Next, we implemented Quadratic Discriminant Analysis (QDA)[10][12]. QDA works by estimating the covariance matrix of each class (anomalies and non-anomalies) and creating a decision boundary between the two classes using a multivariate normal distribution. QDA assumes that our data are normally distributed within each class, and when this assumption holds, can have the benefit of producing more stable parameter estimates than logistic regression [9]. Unlike LDA, QDA does not make the assumption that the covariance matrices are equal among the classes, allowing for more flexibility (non-linear decision boundaries). Like binary logistic regression, QDA is a relatively simple model that is highly explainable and not prone to overfitting.

The CV estimate for the cross entropy loss of our QDA model is 0.9033, with an F1 score of 0.9420. This was by far the poorest performance of the five models we tried, as shown in the table in 4. This could indicate that our classes are not easily separable, or that the assumption of normality within each of our classes does not hold particularly well for these event count matrices.

4.1.3 K Nearest Neighbors

For our third model, we implemented a simple K Nearest Neighbors (KNN) model[10][13]. This simple model works by computing the Euclidean distance between the inputted observation and the observations in the training set, and returning the most frequently observed class among the k closest observations. This algorithm can be sensitive to class imbalance, but makes no assumptions about the underlying distribution of the data (is non-parametric). KNN is highly sensitive to the scale of our features, making the PCA scores a good choice.

Our KNN model with default hyperparameters (k=5) attained a cross entropy loss of 0.1231, and an F1 score of 0.9932. The key hyperparameter to tune in a KNN model is k, the number of nearest neighbors to consider. To do this, we performed exhaustive grid search with values of k from 0 to 50. We found that the optimal value for k (as defined as the value with the lowest CV cross entropy loss) was 1, however, we decided to go with the second best value of k=3. This decision was motivated by the relatively small difference in cross entropy loss between these choices, and the fact that a KNN model with k=1 will have very high variance. Our tuned KNN model achieved a much lower cross entropy loss of 0.0274, and a slightly lower F1 score of 0.9976.

4.1.4 Random Forests

Next, we implemented a random forest classification model[10][14]. Random forests are a type of ensemble model, and generate predictions by fitting a large number of classification trees in which a random subset of the features are considered at each split and taking the average of the predictions returned by all of these trees. This allows them to achieve lower bias and variance in comparison to a standard classification tree, as well as much higher predictive power. This comes at the cost of increased model complexity and therefore decreased explainability.

Our random forest classifier with default hyperparameters achieved a cross entropy loss of 0.0189, and an F1 score of 0.9978. In an effort to improve this, we tuned the criterion, maximum number of features, and the maximum tree depth hyperparameters. The criterion determines what function is used when determining splits, and we tried the gini impurity and the entropy. The maximum number of features parameter controls the number of features that are considered at each split, and includes the square root of the number of features as well as the base 2 log. Finally, the maximum tree depth controls how “deep” each of the trees in the forest are allowed to grow, thereby acting as a way to prevent overfitting, and included values spanning from 30 to 60 in increments of 10 as well as “None”. Given the size of the parameter space as well as the relatively long fit time, we implemented random grid search with 15 iterations, and used the out of bag (OOB) cross entropy loss to assess the performance of each set of hyperparameters

(5-fold CV was again used to assess the final tuned model). This led us to select the entropy criterion, the sqrt of p for the maximum number of features, and 40 for the maximum depth (meaning that only the maximum depth parameter changed from the defaults). These allowed us to attain a slightly lower CV cross entropy loss of 0.0177, and an almost identical F1 score of 0.9978.

4.1.5 Gradient Boosting Classification Trees

For our final model, we implemented a gradient boosting decision tree classifier. Like random forests, this is a type of ensemble model. Gradient boosting classifiers "learn slowly" by iteratively fitting a large number of shallow decision tree classifiers to the residuals, moving closer and closer to the final prediction in an amount that is determined by the learning rate. While any individual tree would attain very poor performance on its own, the iterative process of "boosting" results in a model with high predictive power [15]. Like random forests, these models achieve lower bias and variance and thus better performance in comparison to a standard classification tree. This performance comes at the cost of very high computational complexity: these models take a very long time to fit in comparison to the other models we examined in this paper. Hyperparameter tuning was conducted on a 10% subset of our training data in order to account for this.

Our gradient boosting classifier with default hyperparameters attained a cross entropy loss of 0.0188 and an F1 score of 0.9978, similar to our tuned random forest. In order to improve this, we tuned the number of estimators, learning rate, subsample, and max depth hyperparameters. The number of estimators parameter controls the number of trees that are built ("boosting stages"), and we examined values spanning from 100 to 500 in increments of 100. For the learning rate, we decided to go with the formula $lr = \frac{10}{\#\ estimators}$. Lower learning rates need more iterations to converge to an optimal solution (and vis versa), and this formula allows us to leverage this relationship in order to improve the efficiency of our hyperparameter selection process by excluding what we know are likely to be bad combinations of these parameters from our search space. The subsample hyperparameter controls the proportion of features considered at each stage of the boosting process, and we tried values from 0.4 to 1 in increments of 0.2. Finally, the maximum depth parameter controls the maximum depth of each tree created at each stage of the boosting process, and we included values from 4 to 10 in increments of two. Given the size of the search space and the very high fit time of these models, we implemented 15 iterations of random grid search using only 10% of our training data. This led us to select 100 estimators, with a learning rate of 0.1, a max depth of 8, and a subsampling value of 0.8. From here, we sought to fine-tune the learning rate by performing exhaustive grid search using learning rate values from 0.05 to 0.15 in increments of 0.01. However, we found that the learning rate of 0.1 resulted in the best

CV estimate of the cross entropy loss. In the end, our tuned gradient boosting model achieved a cross entropy loss of 0.016 and an F1 score of 0.9980, both better than any of our other models.

4.2 Final Model Selection and Evaluation

As outlined in section 3.4, we primarily used the CV estimate of the F1 score in order to select our final model. Overall, the most striking part of our results is how well all of the models that we implemented performed. Every one attained a CV F1 score of above 94%, and all but the QDA model were above 98.9%. There was somewhat greater variation in the CV weighted binary cross entropy loss, with the QDA model having a value of 0.9, the logistic regression and default KNN models sitting in the range of 0.11 to 0.13, and the rest of the models sitting in the range of 0.016 to 0.0278. The CV accuracy estimates ranged from 0.9965 to 0.9999, though as explained in 3.4, accuracy is not robust to the severe class imbalance present in this dataset. These results suggest that anomaly detection on HDFS logs using the event count matrices is highly feasible, and even relatively simple models are able to detect anomalies with a very high degree of accuracy. More complex tree-based models achieve astonishingly high results, with precision and recall scores well above 99.5%.

As shown in section 4, our tuned gradient boosting classification model achieved the highest CV F1 score, with a value of 0.9980 (the second best, other than our gradient boosting model with default parameters, was our random forest, with a very close 0.9978). **It also achieved the lowest CV weighted binary cross entropy loss** at 0.0163 (compared to 0.0189 for our random forest). In order to assess the expected real-world performance of this model, we examined its performance on the test set that was withheld at the beginning of our experiment. Because this data was not used for the purposes of training, hyperparameter tuning, or model selection, it offers the most unbiased way of assessing this. The results are reported below:

Weighted Binary Cross Entropy	Accuracy	Recall	Precision	F1 Score
0.004872	0.999872	0.998401	0.997206	0.997803

Here, we see that our model achieves very high performance across the board, having values that are in line with those observed from our 5-fold cross validation. While this model would likely perform extremely well in production, further research could seek to make marginal improvements by implementing more complex models such as extreme gradient boosting classification trees or deep learning methods such as neural networks.

REFERENCES

[1]Pinjia He, Jieming Zhu, Shilin He, Jian Li and Michael R. Lyu. An Evaluation Study on Log Parsing and Its Use in Log Mining(https://pinjiahe.github.io/papers/DSN16.pdf)

[2] R.Vaarandi,“Adata Clustering Algorithm For Mining Patterns From Event logs,” in IPOM’03: Proc. of the 3rd Workshop on IP Operations and Management, 2003

[3] A. Makanju, A. Zincir-Heywood, and E. Milios, “Clustering event logs using iterative partitioning,” in KDD’09: Proc. of International Conference on Knowledge Discovery and Data Mining, 2009.

[4]Q. Fu, J. Lou, Y. Wang, and J. Li, “Execution anomaly detection in distributed systems through unstructured log analysis,” in ICDM’09: Proc. of International Conference on Data Mining, 2009

[5]L. Tang, T. Li, and C. Perng, “LogSig: generating system events from raw textual logs,” in CIKM’11: Proc. of ACM International Conference on Information and Knowledge Management, 2011, pp. 785–794.

[6] Adetokunbo Makanju, A. Nur Zincir-Heywood, Evangelos E. Milios, “Clustering Event Logs Using Iterative Partitioning”.

[7]Shilin He, Jieming Zhu, Pinjia He, Michael R. Lyu. Experience Report: System Log Analysis for Anomaly Detection, *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2016. [Bibtex][中文版本](ISSRE Most Influential Paper)

[8] H. Mi, H. Wang, Y. Zhou, R. Lyu, and H. Cai. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud com- puting systems. IEEE Transactions on Parallel and Distributed Systems, 24:1245–1255, 2013

[9] http://uc-r.github.io/discriminant_analysis
[10] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011
[11] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[12]https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html

[13]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[14]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[15] https://bradleyboehmke.github.io/HOML/gbm.html
[16] J. H. Bellec and M. T. Kechadi. CUFRES: clustering using fuzzy representative events selection for the fault recognition problem in telecommunications networks. In PIKM ’07: Proceedings of the ACM first Ph.D. workshop in CIKM, pages 55 – 62, New York, NY, USA, 2007. ACM.
[17] Wei Xu, Ling Huang, Armando Fox, David Patterson, Micheal I. Jordan, “Detecting Large-Scale System Problems by Mining Console Logs”, 2009