ANALYZING BOARD HEALTH WITH MACHINE LEARNING

A Major Qualifying Project Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the Degree of Bachelor of Science.

Authored By

Suverino Frith

Josh Lovering

Clay Oshiro-Leavitt

Bryson Tang

Advised By

Kyumin Lee
kmlee@wpi.edu

Sponsored By

Renee Walker
Renee.Walker@dell.com

Alex Sanville
Alex.Sanville@dell.com

ABSTRACT

The goal of this project was to determine the reason for unexpected test failures through analysis of Dell EMC motherboard testing log data. We began with an extensive exploration of our extracted features, through a series of statistical analysis and plotting. Our solution consisted of three components: a warning system to detect trend changes in feature values, a machine learning model to predict failure tests, and the standardization of log data to increase efficiency of future analysis. The warning system analyzes a group of test logs using time series and regression analysis and outlier detection to calculate the date at which future tests would reach a mean value equal to an outlier value of the current tests. The machine learning model was trained on 548 features from a set of 826 passing and 511 failing boards to classify unlabeled data, and received a testing accuracy of 63 percent. Lastly, we contributed a standardized log format proposal based upon JSON which would increase the value of their log data. This was determined after the extensive time required for parsing the different formats across test steps during feature extraction.

ACKNOWLEDGMENTS

We would like to thank the team from Dell EMC for their ongoing support and help throughout the entire project. Without their continual feedback, insight, and work on our behalf, our project would not be nearly as successful. We would also like to thank our project advisor, Professor Kyumin Lee, for his help and guidance.

EXECUTIVE SUMMARY

In this project, we applied machine learning algorithms and designs to a specific portion of the Warnado product lifecycle from Dell EMC. The Warnado motherboards undergo system testing before widespread launch and use. Dell EMC identified several shortfalls in their testing regimen with regard to their handling and analysis of motherboard log data. To remedy these issues, we researched, designed, and implemented a machine learning pipeline to parse motherboard log data into usable features and apply machine learning models to the data. The models can accurately determine whether a board has failed or not and provide useful debugging information such as predictable trends in data.

We approached this project by using the data science process - through which, we explored the raw data, identified useful features and components, reduced our feature set, performed machine learning trials, and analyzed trends and patterns to make conclusions. Our learning experiments had two main components - classification and trend detection. Using various machine learning models such as support vector classifiers and clustering, we achieved a classification accuracy of 63 percent on a sample of 1,337 motherboards from two different test versions. For trend detection, we created a warning system application that performs trend analysis and detection across log entries. This tool will help Dell EMC determine deviations from manufacturing specifications and tolerance slippage.

We were able to identify some key factors that would improve Dell EMC's workflow. The final factor was standardizing the log data format, which would ready Dell EMC for the future of big data and AI by having structured data that can easily be fed into a machine learning model. This will save future data scientists weeks of data cleaning and feature extraction, allowing the process to be expedited. The machine learning model serves as a tool for which Dell EMC can use to save testing time on boards with inevitable failures. The warning system enables Dell EMC engineers to easily monitor motherboard health trends. This exploratory project gave the Dell EMC team

a better understanding of their motherboard log data, and will serve to increase the efficiency of future projects.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

As computing has continually increased in power, computation and automation have been used in more and more scenarios to enhance productivity. Advances in artificial intelligence, specifically in machine learning, have propelled expanded use of these technologies in manufacturing and product development. In this project, we applied machine learning algorithms and designs to this domain - using machine learning to classify and detect failing motherboards in product testing.

Our project focused on a specific portion of the Warnado product lifecycle from Dell EMC. The Warnado system is currently undergoing system testing before widespread launch and use. However, Dell EMC had identified several shortfalls in their testing regimen. When a Warnado motherboard fails, determining the cause of failure is time consuming. There is no automated pipeline for failure or trend analysis. To analyze a specific motherboard failure, multiple log files need to be sourced, collated, and reviewed by hand to determine the root cause of failure. This manual review process is time intensive, and the granular level of analysis results in larger trends being missed.

Automation of this analysis has been difficult due to the structuring of the log data itself. The information for each board is spread between numerous files, and the format of the data itself varies. While the data loosely follows a JSON format, individual features have wildly different data representations - some are integers, others are entire paragraphs of text.

Our goal was to solve these shortfalls in Dell EMC's process with new classification and analysis tools for determining failures and trends in their motherboard tests. We researched, designed, and implemented a machine learning model that can parse the motherboard log data into usable features for machine learning applications, accurately determine whether a board has failed or not, and provide useful debugging information such as predictable trends in data.

Upon completion, we implemented a robust data processing pipeline that achieved a classi-

fication accuracy of 63 percent on a sample of 1337 motherboards from two different test versions as well as created an application that performs trend analysis and detection across log entries. This tool will help Dell EMC determine deviations from manufacturing specifications and tolerance slippage.

2 Sponsor Description

Dell EMC is the result of the 2015 merger between technology companies Dell and EMC. Under the Dell EMC brand, they specialize in data storage, networking, information services, virtualization, analytics, and cloud computing.

For our project, we worked alongside the PowerStore Platform and System Engineering team, based out of Hopkinton, Massachusetts. This specific team focuses on designing and rapidly deploying new hardware platforms and systems for industry and enterprise use.

3 Background

To begin our work, we first turned to information on the Warnado data itself as well as existing research and methodologies to determine possible paths forward.

3.1 Warnado

The Warnado boards are part of the Dell EMC PowerStore X family of data storage products. These products are designed as a scalable data solution, focusing on enhanced resource utilization and performance to allow for rapid growth and flexibility. Inside a PowerStore cluster, it is possible to scale storage by installing or removing additional drives.

3.2 Machine Learning and Analysis Methods

Machine learning is a common method of data analysis and prediction within the field of artificial intelligence. The following section will give an overview of common machine learning designs and uses.

3.2.1 Multiple Instance Frameworks

Multiple instance frameworks (or multiple instance learning/MIL) refer to the set of machine learning methods that utilize bagging in labeling. In MIL, class labels are defined on bags (sets) of data - all instances in a negative bag are truly negative, while positive bags may contain false positives.

3.2.2 Support Vector Machines

Support vector machines (SVM) are a commonly used method for binary classification in machine learning. In the classification field, a subset of SVMs are used - these are known as support vector classifiers (SVC). A support vector machine works by finding a hyperplane that separates two classes. The location of the hyperplane is optimally defined by a series of vectors -

these vectors 'support' the plane. In a SVM, loss is defined as the penalty for misclassification of an element - this can be further refined in terms of margin. A hard margin refers to a SVM that has no misclassifications. A soft margin allows for some misclassifications. Depending on the data, there may or may not be a possible hard margin classifier.

For SVMs, hard margin loss classifiers provide better generalization performance than a traditional ramp loss SVM classifier model. Hard margin loss models also have greater performance, especially for large data sets. (Poursaeidi, Kundakcioglu, 2014)

Support vector machines can be further enhanced through the creation of multiple instance SVMs (MISVM). MISVMs treat bagged data as a singular classification event, rather than considering each point in a bag as a separate element (Cano, Melki, Ventura, 2018). Here, the model predicts on each instance using a SVM and then trains the model to tell the instances' classes, thus using the multiple instance framework to provide classification (Murray, Hughes, Kreutz-Delgado, 2005).

Another approach to SVM data processing is spectrum representation (Fulp, 2008). With spectrum representation, each observation is given each single-value representation; this "tag" value represents the specific observation in a k-length sequence of observations. This labeling results in bk possible sequences, where b is the number of different tag values. Here each sequence is assigned a unique value f, which then becomes a feature value. The frequency of each sequence in the k-length sequence of observations is then used to create a vector, where every value is sequence-value:count. This vector is then used in the support vector machine for classification.

There are multiple adjustments that can be made to the spectrum representation method. Additional features can be incorporated as well, such as observation times and observation content. However, adding more information must be balanced with sequence length as the number of features grows exponentially.

3.2.3 Outlier Detection

Outlier detection (also known as anomaly detection) is the identification of data points which differ significantly from the majority body of data. While this can be done in multiple ways, this section will focus on one particular algorithm known as isolation forest.

Isolation forests are an anomaly detection algorithm that randomly selects and partitions features until they are isolated. Once this has been performed the outliers in the data will be distinguishable by residing in low density regions (Scikit-Learn, 2020). This attribute enables isolation forests to have high accuracy and maintain performance even when dimensionality increases (Liu, Ting, Zhou, 2008).

3.2.4 Time Series Analysis

In machine learning, time series analysis has been used for trend and outlier detection. A time series is a series of data organized by its timestamp or point of collection. Temporally organized data lends itself well to trend detection (NIST, 2020).

Trend detection is a common analysis for time driven data. To get an initial overview of the data, one of the most common approaches is to graphically plot the data. Once plotted, significant trends can often be spotted by eye. To detect more nuanced trends, regression functions are often overlaid onto the data. These regression plots both illustrate trends, as well as define equations that model these trends.

There are further ways to analyze time series. Another more advanced approach to time series data analysis is through a lag analysis. Normally, models consider the correlation between consecutive data points. However, there can be trends which are offset, or 'lag' in the data. These can be determined through a lag plot, where data points are measured against data sampled at an earlier time. The shape of this plot can be used to determine autocorrelation - whether data is correlated based on a time difference.

Lastly, one can perform stationarity analysis on time series - whether the data of a time series is dependent on the time of collection. The Augmented Dickey-Fuller Test is typically used for this

calculation. This test checks whether a unit root is present for the time series - a core characteristic of non-stationarity. If the test passes, the time series has some form of trend. Otherwise, the time series is truly stationary and there is no trend in the data.

3.2.5 Feature Selection

Feature selection is critical for creating workable datasets from large, sparse collections of data. These algorithms are used for removing irrelevant, redundant, or otherwise negligible features from the dataset to remove unneeded dimensionality. High dimensionality can result in numerous problems in machine learning applications - too many features can result in the overfitting of a model, therefore reducing performance in testing. High dimensionality can also diminish the benefit of clustering - as more features are added, distances between observations begin to equalize. This phenomenon is commonly known as 'the curse of dimensionality'.

3.2.5.1 Chi Squared

Chi Squared is a statistical test to identify the correlation between two sets of samples or if they are independent from each other. It can be applied to feature selection as each feature is compared to the target variable to test for independence. (Jin, Xu, Bie, Guo, 2006) Once the correlation between each feature and variable has been calculated using equation 3.1, the features with the highest correlation value are selected. This will ensure that the features being used will be the most correlated features with the target variable.

$$X^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Where O_i = Observed Value and E_i = Expected Value

Figure 3.1. Chi Squared Calculation

3.2.5.2 Mann-Whitney

The Mann-Whitney/Wilcoxon Signed Rank Test is a nonparametric test of the null hypothesis that two samples are drawn from different populations (Mann, 1947). It returns a p-value that roughly represents the probability of sampling again and getting different results.

3.2.5.3 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised feature selection method that reduces a dataset to a set of representative, variable data. This set is produced through the calculation of the principal components - the product of each weighted feature in the feature set. Through the calculation of these components, PCA finds a representation of the dataset with low dimensionality while still explaining the majority of the variance. This process removes redundant correlated features as well as features that have little impact on the variance of the dataset (James, Witten, Hastie, Tibshirani, 2013). Principal component values are calculated in order of decreasing feature variance. The first principal component is calculated as the normalized linear combination of features following the following equation:

$$Z_i = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p$$

$$\text{where } \sum_{j=1}^{p} \phi_{j1}^2 = 1$$

Figure 3.2. First Principal Component Calculation

3.2.6 Previous Work

Previous work in the field of hardware malfunction detection has shown that machine learning is a successful technique (Gonfalonieri, 2019). Predictive maintenance is the field of understanding when a piece of hardware is in need of maintenance by utilizing machine learning. By allowing a computer to make decisions, the hardware can become scaliable. A human has to understand the warning signs and keep an eye on each piece of hardware, whereas a machine learning algorithm can watch over all of them. Another approach is scheduled maintenance, but it can be wasteful if preemptive, and catastrophic if done too late (Sipos, Fradkin, Moerchen, and Wang, 2014). The machine learning algorithm can give a warning when it predicts a piece of hardware will fail and a human can assess the physical situation and make the required repairs. There are many techniques to develop a machine learning algorithm capable of this and this section will explore the process to creating one. There are three important steps that take place during a predictive maintenance project. Having the right data available, framing the problem appropriately, and evaluating the predictions properly (Gonfalonieri, 2019) .

When collecting data it is important to have both the data scientist and the domain expert involved. The data should be collected over the life of the hardware and capture its deterioration process. It's important to understand when and what maintenance might have taken place over this time frame as well. When collecting data there are a few questions that should be asked.

- Which question do we want the model to answer?
- Is it possible with the data we have at our disposal?
- Is the failure a sudden event, or is there a slow decline before complete malfunction?
- Is every recorded event labelled?
- When labelled events are available, what is the proportion of the number of events of each type of failure and events of well functioning data?
- How long in advance should the model be able to indicate that a failure will occur?

- What is the consequence of not predicting a failure or predicting a failure that will not happen?
- Which components are typically associated with this type of failure?
- Which parameters should be measured that most signify the state of component/machine health?
- What is the required accuracy and frequency of the measurements needed?

Once the data has been collected, it is important to frame the problem correctly. This will come with feature engineering, as the type of information contained in the data is revealed, the goal of the model should be evaluated. Two models that can be trained on the data are a regression model for the remaining useful lifetime and a classification model to predict failure within a given time window. The regression model will need labeled data that describes the remaining useful life (RUL) of the piece of hardware (Liu, Hu, and Jin, 2019). Transfer learning utilizing a LSTM model can be an effective model to predict the RUL of a piece of hardware at any point in it's life. For classification, the different types of failures must be known and the model will classify the piece of hardware with the error likely failure to occur (Gonfalonieri, 2019). This has the advantage that the exact failure can be pinpointed and handled, where using RUL only gives an estimate of how much longer the piece of hardware has. The disadvantage of a classification approach is that all the failure types must be known and labeled. When there are failures that happen sparsely it could be difficult to have knowledge of them. The RUL does not need to have that knowledge, just the end of life date. When training the models, the data can be split into and trained on time chunks in order to represent a board that is only part way through its life. Once the model has been developed and implemented into the real world, the next step is considering where to add additional sensors. This can be done by understanding where the features have blind spots and deriving new measurements that could handle those shortcomings.

A multiple instance learning (MIL) model has also proven accurate when dealing with predictive maintenance. In preprocessing the data for an MIL model, the data is split into bins that

are equal time periods or have equal frequency of data points. These equal data bins are known as instances. Binning the data has the added benefit of dimensionality reduction and overfitting being reduced (Murray, Hughes, Kreutz-Delgado, 2005). With MIL, if there is a failure in one instance, the whole piece of hardware is considered a failure (Murray, Hughes, Kreutz-Delgado, 2005). Thus, to train a model each instance is input and trained to classify if the instance is a failure or not. A model that has shown great success in classifying instances is SVM (Murray, Hughes, Kreutz-Delgado, 2005). For SVMs, hard margin loss classifiers provide better generalization performance than a traditional ramp loss SVM classifier model. Hard margin loss models also have greater performance, especially for large data sets (Poursaeidi, Kundakcioglu, 2014). When designing a SVM for MIL there are three questions that should be considered.

- For the set of consistent classifying hyperplanes, which MISVM encodes a feasible zero loss solution for each consistent hyperplane?
- Do all feasible solutions with zero loss correspond to a consistent hyperplane?
  - This is known as soundness
- Are the corresponding feasible solutions convex?

In practice, no MISVM can fulfill all three features. It is important to identify which features to prioritize in development (Doran, Ray, 2014).

4 Methodology

To begin our work, we first turned to information on the Warnado data itself as well as existing research and methodologies to determine possible paths forward.

4.1 Iterative Development

Throughout this project, we utilized an iterative approach for our design and deliverable creation. Iterative design allows for constant revision and growth, focusing on short development cycles with a functional design at the end of each cycle. By keeping development cycles short, we were able to solicit feedback from Dell EMC and our advisor weekly and incorporate it into our deliverables for the following week. In addition to adding new features, we continually explored data features and implemented them in our learning model. This allowed us to continually increase the accuracy and performance of our machine learning pipeline.

This process broken down into several distinct phases (see Figure 4.1). The first of these steps is requirements gathering - during this time, criteria are developed for the project and deliverables. The end product is ideated based on requirements while considering design and practical constraints. The next stage is planning. Here, ideas are developed to allow for a thorough and measured approach to development. Design, implementation, and evaluation are the next portion of the process. Plans are turned into concrete architectural decisions and are implemented in software. Evaluation and feedback from sponsors ensure that the deliverable is meeting expected functionality and use. At this point, we consider whether or not the deliverable is capable of performing all required tasks - if so, the deliverable is considered complete and the design is finalized. Otherwise, the process begins again by returning to the planning phase.

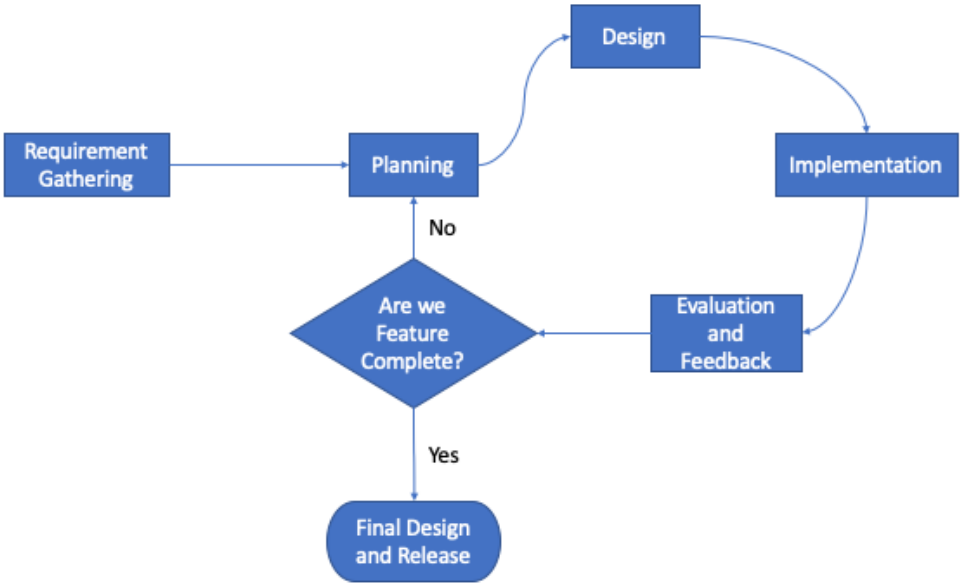

Figure 4.1. Illustration of the Iterative Design Process.

4.2 Data Science Process

In the field of data science, there is a general process used when approaching a problem. This process has the following steps: data exploration, feature extraction, feature selection, model development, and analysis of results (see Figure 4.2).

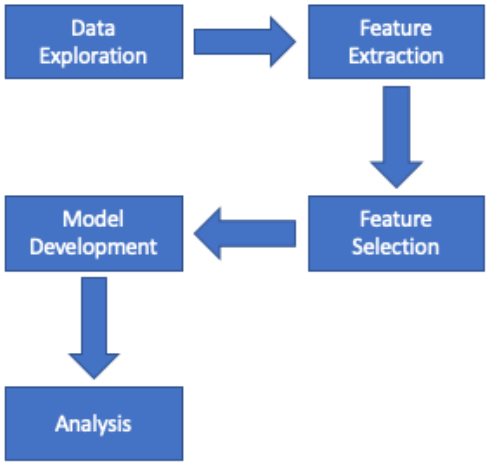

Figure 4.2. Flow of the Data Science Process.

4.2.1 Data Exploration

The first step is data exploration. During this phase, we considered various questions that would help us gain a deeper understanding of the provided data and what it can help us accomplish. These questions are the following:

- Which question do we want the model to answer?
- Is it possible with the data we have at our disposal?
- Is the failure a sudden event, or is there a slow decline before complete malfunction?
- Which components are typically associated with this type of failure?
- Which parameters should be measured that most signify the state of component/machine health?

To answer these questions, we created a series of data explorations to look for trends and notable patterns in the data. For this task, we used Jupyter notebooks. This allowed for rapid development

and prototyping, as notebooks allow for individual methods to be executed, doing away with the need to write a more advanced UI for these initial tasks. In these explorations, we calculated summary statistics of the features' datasets as well as created visuals to display variation, skews, and general pattern distribution. This enabled us to find relevant features that were valuable for our future work and development. See Appendix A for graphs showing variance in features.

4.2.2 Data

Our data came in the form of log files for the Dell Warnado product line. Each board had a directory of associated JSON files, containing test information, metadata, identifying information. Log formats changed between testing versions, with our samples split between Dell's v14 and v15 testing standards. Our data consisted of 780 v14 logs and 595 v15 logs, with a grand total of 1,375 logs. However, only 1,337 logs contained at least 50% of the features. Those additional 38 logs were discarded.

Working with the two testware standards proved to be difficult. Between the two versions, formatting for test results changed as well as some tests being deprecated and new ones added. Similarly, testing information being spread between multiple files required us to compile information from multiple sources for feature extraction.

To process the log data, we wrote a series of Python scripts to convert the raw data from log files into workable formats for our learning pipeline.

4.2.3 Feature Extraction

To implement the feature extraction, we wrote a series of Python scripts for each test step in which we found significant variance, data skewing, or other non-gaussian distribution. Specifically, we were interested in differing distributions between passing and failing boards. Each script was designed to write the selected features out to CSVs indexed by board UutId and whether that board was a pass or failure. These extraction scripts were run over all board samples.

Features were selected from the following test steps:

- Drive_max_temp_check

- Drive_media_io
- Drive_performance_test
- Heath_check
- Hw_check
- Slic_check
- Test_step

Through our feature extraction algorithms, we had identified and extracted 548 distinct features from the version 14 and 15 board logs.

4.2.4 Feature Selection

For feature selection, we implemented three common methods of statistical feature selection. These feature selection methods were vital for reducing unnecessary data dimensionality. For the project, we implemented and trialed the following three methods:

- Chi^2
- Mann-Whitney
- Principal Component Analysis (PCA)

For more information on these feature reduction algorithms, please consult section 3.2.5. Below are results from feature selection algorithms PCA and Mann-Whitney (See Figure 4.3 and 4.4).

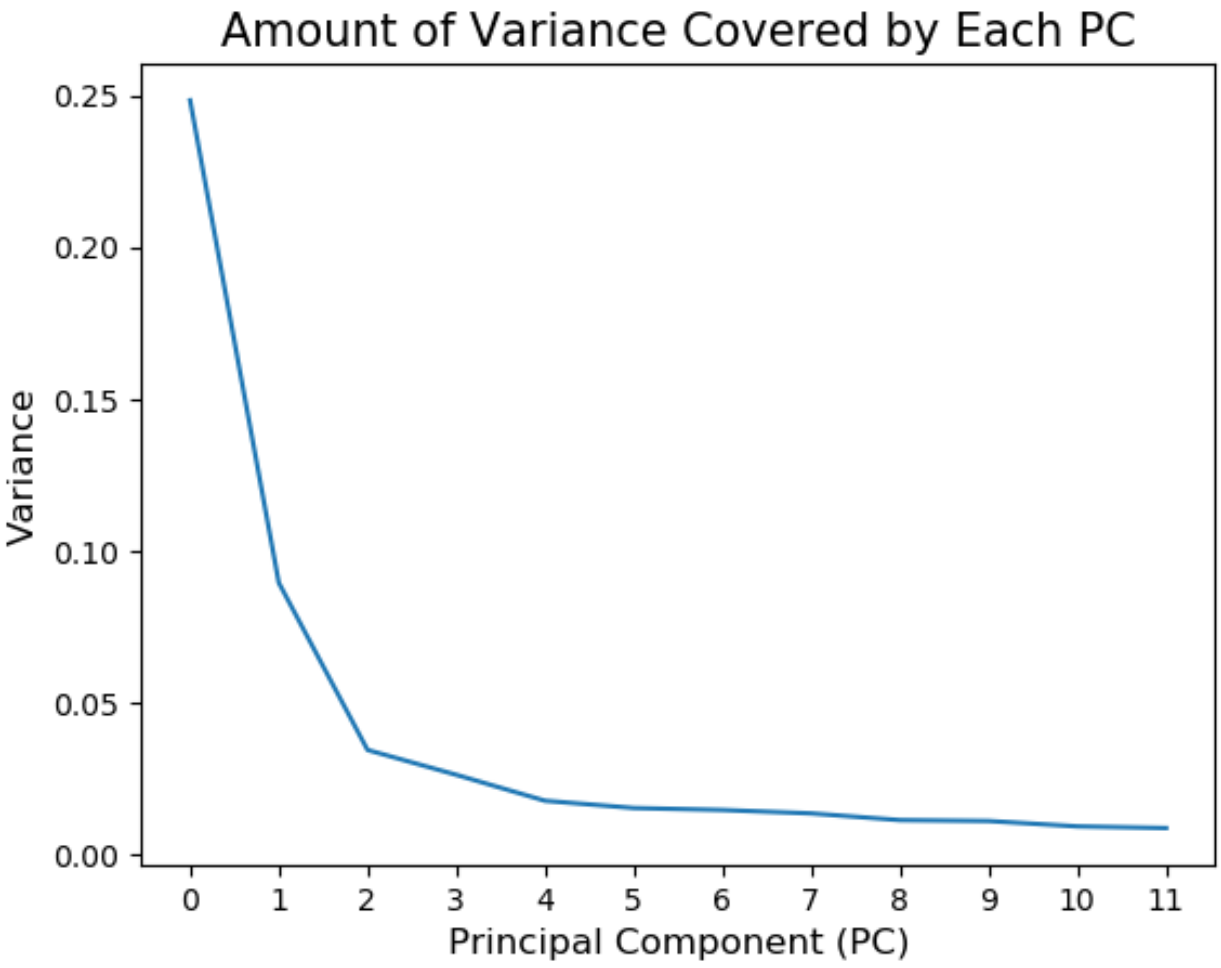

Figure 4.3. PCA Variance Ranking by Principal Components

	Feature	Mannwhitneyu	rank
0	SurvivedMemorySize-BootMon	0.009982	6.0
1	RunTime-USB_Test	0.009982	6.0
2	TotalAllocatedMemorySize-Drive_PopulationCheckTest	0.009982	6.0
3	SurvivedMemorySize-USB_Test	0.009982	6.0
4	RunTime-CaptureLogs	0.009982	6.0
5	RunTime-Front_LED_Check	0.009982	6.0
6	RunTime-Drive_AutoDump_Check	0.009982	6.0
7	TotalAllocatedMemorySize-Discovery	0.009982	6.0
8	SurvivedMemorySize-Drive_TestUnitReadyNoError	0.009982	6.0
9	RunTime-Drive_Format	0.009982	6.0
10	TotalAllocatedMemorySize-PowerOff	0.009982	6.0
11	SurvivedMemorySize-RegisterOSInstall	0.018273	12.0
...	...	...	...
400	RunTime-ArrayInfo	0.451262	357.5
401	RunTime-Ready_Linux_Boot	0.451262	357.5
402	RunTime-CMD_Check	0.451262	357.5
403	TotalProcessorTime-Ace-_Disc	0.500000	408.0
404	TotalProcessorTime-Drive_Performance_Test	0.500000	408.0
405	TotalProcessorTime-Set_Temp_PXE	0.500000	408.0
406	TotalProcessorTime-Retired_Die_Check	0.500000	408.0
407	SurvivedMemorySize-Drive_Pattern_Media	0.500000	408.0
408	SurvivedMemorySize-Drive_Firmware_Check	0.500000	408.0
409	TotalProcessorTime-Drive_Format_Check	0.500000	408.0
410	SurvivedMemorySize-Ace-_Gene_(Nested)	0.500000	408.0
411	TotalProcessorTime-Drive_Firmware_Check	0.500000	408.0

412 rows × 3 columns

Figure 4.4. Mann-Whitney Feature Rankings

After finding significant data features, we tested possible data transformations to discern clearer relations between the feature and whether the board was a failure or not.

4.2.5 Model Development and Tuning

After feature extraction and feature selection methods are performed, it is time to turn to model development, training, and tuning. Leveraging the Python library 'Scikit Learn', we were able to create several different classification models for our project and compare model accuracies. The following models were created:

- Support Vector Classifier
  - Linear
  - Radial Basis Function
- Random Forest Classifier
- Naïve Bayes Classifier
- XGBoost Classifier

For these models, we used a 70/10/20 data split. This left 70 percent of our data set for model development and training, 10 percent of the data for model validation, and the remaining 20 percent was used for model testing.

4.3 Time Series Analysis

During our exploration of the data, the Dell EMC team presented us with a recent issue they had faced. There were a growing number of motherboards failing within a battery test. The failures were due to batteries showing voltages below their pass/fail criteria. Upon further investigation through time series analysis, it was discovered that their boards had been trending towards lower voltage values for the past few months. This sparked a plan to develop a warning system in order to prevent surprises like this one. Time series analysis could be used to monitor motherboards over time and warn the Dell EMC team if there are trend changes.

We visualized the voltage trends through scatter plotting. The x-axis would show the date of each board's battery test, while the y-axis displays the voltage. This plotting was functionalized, in order to be used within our warning system. Using the feature extraction script established earlier in our project, a plot could be generated for each feature on an entire batch of logs.

Through the use of linear regression, a trendline could be displayed over the plot. The severity of this line would indicate the need for a given feature to be warned of. If the values for a feature are increasing or decreasing over time, Dell EMC, may want to investigate the source. Depending on the cause of the change in trend, either the pass/fail criteria or method of performing the given test may need to be adjusted.

4.4 Outlier Detection

We used outlier detection in order to isolate any anomalous boards within a given set of motherboard testing logs. For this, we took advantage of the isolation forest (iForest) algorithm, which locates anomalies within a dataset. The inlier range generated by iForest was overlaid over the regression analysis for a set of test logs. This range serves to separate expected board values from anomalous ones, for which investigation is recommended. When a trendline passes across this threshold, the given feature would be flagged with a warning for the Dell EMC team to investigate. When isolating anomalies, we used a low contamination value of 0.15 - 15 percent of values in the dataset will be considered outliers - in order to prevent preemptive warnings that could result in inefficient use of company time.

We picked the isolation forest algorithm due to its recorded success with large datasets. Isolation forest is powerful with high dimensionality, especially where there are potentially irrelevant features (Liu et al., 2008). Due to the nature of our application, this would be applicable. We extracted as many features as we were able to during our allotted time, without the domain knowledge to determine their value or redundancy. While we explored dimensionality reduction methods for our machine learning application, we would be keeping the entire dimensionality for this portion of the project. We will be monitoring all of the features we extracted from every portion of

the testing procedure, as we have no reason to monitor only a subset of the test steps for changes within trends.

Isolation forest has low processing time and does not require as much memory as other, more traditional methods (Liu et al., 2008). This is due to the shorter tree paths required when isolating anomalies, which are naturally separated within a dataset. This efficiency is advantageous for us as Dell EMC users will manually be running the final program.

While this method is known for a high accuracy within global anomalies, it is not well suited for anomalies that exist within the global inlier range (Gao et al., 2019). Fortunately, due to our application of this algorithm, we will not be affected by this restraint. We are utilizing isolation forest specifically to generate a range of expected values, and monitor trends within test steps that begin to formulate outside of this boundary. For the purpose of monitoring the trends of entire groups of motherboard test logs, we will not be focusing on local anomalies. Local anomalies require multiple attributes to detect, and would not fit in with this trend change warning system.

4.5 Frameworks and Resources

To implement our design, we needed a versatile language capable of both exploration and software programming. For this, we selected the incredibly popular and powerful language Python.

4.5.1 Python

There are multiple reasons why Python was chosen for this project. Due to its scripting and interpretive nature, Python lends itself well to writing short, task oriented programs that can be used for data exploration and extraction. These can then be combined to form larger portions of our design.

Python is also incredibly popular in the data science and artificial intelligence fields. Due to this, there are a multitude of packages and toolkits freely available to aid in our work. Some of the most notable packages we used were: Pandas, Scikit Learn, and Seaborn. We have been exposed to these libraries in our academic backgrounds, and have read about them used for industry applications.

4.5.1.1 Pandas

Pandas is a common library used for working with large quantities of data. It contains mainly critical functions used for easily reading a variety of different data formats, preprocessing, and manipulating data fields. This library was built on top of the Python NumPy library, which is a standard library used for matrix manipulation. Pandas adds high level functionality, which makes preprocessing data for analysis and AI applications elegant and straightforward.

4.5.1.2 Scikit-Learn

Scikit-Learn is a popular artificial intelligence and machine learning library built for Python. Scikit-Learn contains functionality for preprocessing, dimensionality reduction, and many modeling techniques. This enabled us to focus on gathering results and testing, rather than reimplementing machine learning and statistical models ourselves.

4.5.1.3 Seaborn

Seaborn is a graphical data visualization library built for Python on top of the existing Matplotlib library. Seaborn allows for advanced graphing of data with simplified commands, reducing time spent on actual graphic generation. We used this throughout our project to functionalize visualization for both exploring our data to generate questions, as well as presenting our results and findings.

4.5.2 WPI Turing

Turing is WPI's primary research cluster used for cloud computing by both faculty and students. The cluster contains a total of 48 servers, comprising thirteen hundred CPU, over nine terabytes of RAM, and 64 GPU. We utilized turing in order to run our preprocessing scripts on large batches of motherboard test logs.

4.6 Design

4.6.1 Feature Extraction

Given a ZIP archive of log files, the Python script will extract the features and create a CSV (comma separated values) file that can then be handled by classification or the trend warning system. First, the ZIP file is extracted in parallel to get the log file for each board. Then, every metadata log file, such as UutDataLog.json and TestStepHistory.json are collected from each board and concatenated to create individual data frames for each file. Once the data frames have been created, the feature extraction scripts run. UutDataLog.json provides features based on values from within the test steps. There are various custom functions that will extract individual features from specific test steps and there is a general function that will extract features that are in the proposed format that will be explored later. The general features are found by searching the output column to find the desired format, extracting the features, and checking which test step they are located. If the table contains more than one row instead of just collecting the value of the column, the max, min, standard deviation, and mean of the columns will be collected and turned into features. TestStepHistory.json provided the time each test step took to run. The test step run times were extracted into features. Every feature that is extracted is put into a CSV, and once all the features are extracted they are combined into one final data set. The format of this data set is that each row represents a board keyed with a unique UutId and the columns are the extracted features. Once the features have been extracted classification and trend detection can be run. The final data set is saved as a CSV file. The flow of data during this program execution can be seen in Figure 4.5.

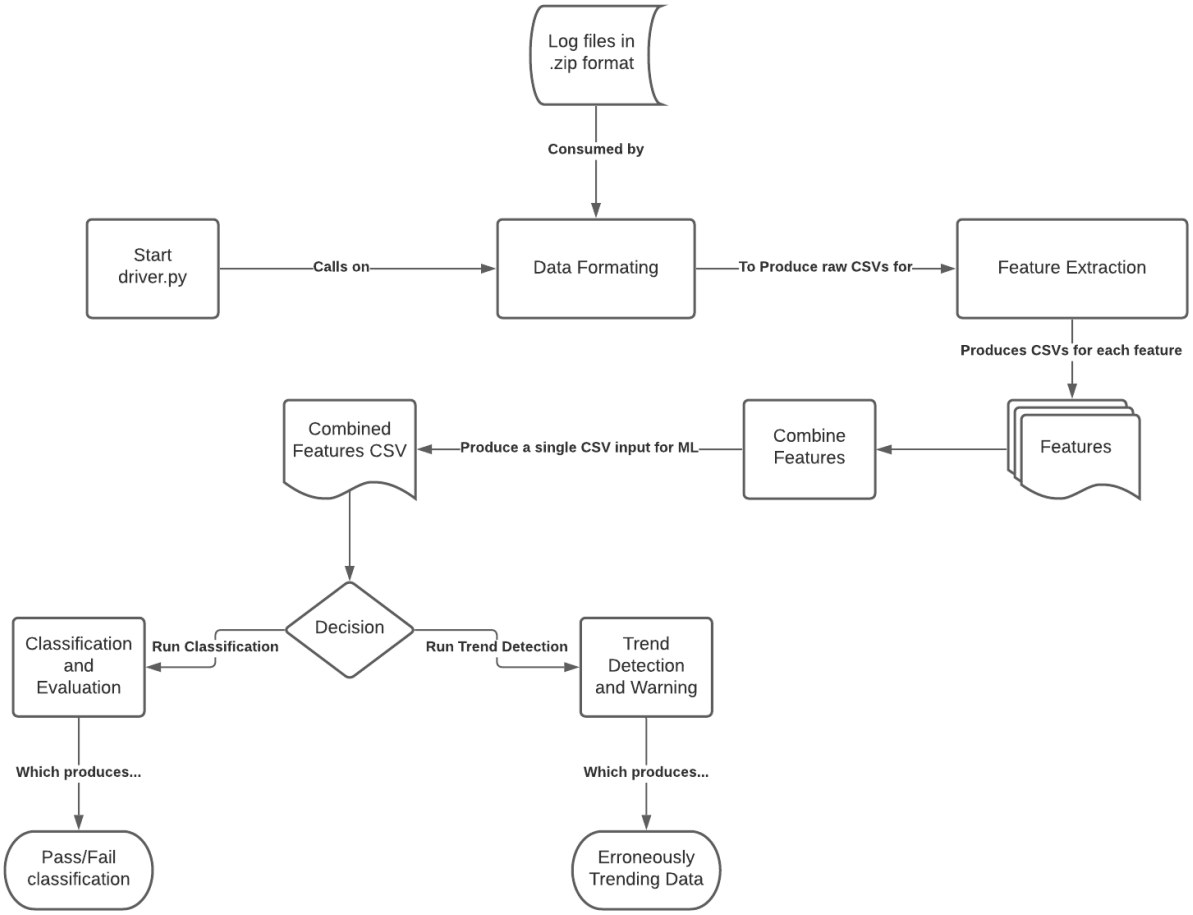

Figure 4.5. Data Processing Pipeline

4.6.2 Classification

After the data has been extracted into CSV, the classifier methods can be utilized. The target for the classifier is whether or not the board failed the tests. The classification scripts compare different models andt parameters to get the best set of parameters for the right model. The script outputs an accuracy report of each model and the best parameters. The script can also take the input of a feature selection method. The types of feature selections that can take place are chi-squared, principal component analysis, and Mann-Whitney test. Along with the feature selection technique the number of features can be input into the script. This will pick the top number of features specified by the user for the model testing. Once the features have been reduced, the 5

machine learning techniques will be run: support vector classifier, k nearest neighbors, naive bayes gaussian, random forest, and gradient boosting. A grid search is performed on each model in order to find the best set of hyper parameters. The hyper parameters can be found in Figure 4.6.

```
classifiers = {
    "SVC": {
        'kernel': ['linear', 'rbf', "poly"],
        'C': [1, 10, 100, 1000]
    },
    "kNN": {
        "n_neighbors": [3, 5, 7, 9, 11, 15],
        "weights": ["uniform", "distance"],
        "metric": ["eucliden", "manhattan"]
    },
    "NB_gaussian": {},
    "RF": {
        "max_depth": [80, 90, 100, 110],
        "max_features": ["auto", "sqrt", "log2"],
        "n_estimators": [100, 200, 300, 1000]
    },
    "XGBoost": {
        "max_depth": [3, 5, 10],
        "min_child_weight": [1, 3, 6]
    }}
```

Figure 4.6. Hyperparameters to be tuned for each model.

Once the best set of parameters has been found and the training is over, the results are exported to a csv. Along with the model and feature selection technique, the results of running the model on a testing set is recorded and the ground truth labels are included. This csv file will then be run through the evaluator script. This script will calculate the accuracy, precision, and recall of the model, alongside the values for a confusion matrix. This information will then be stored in

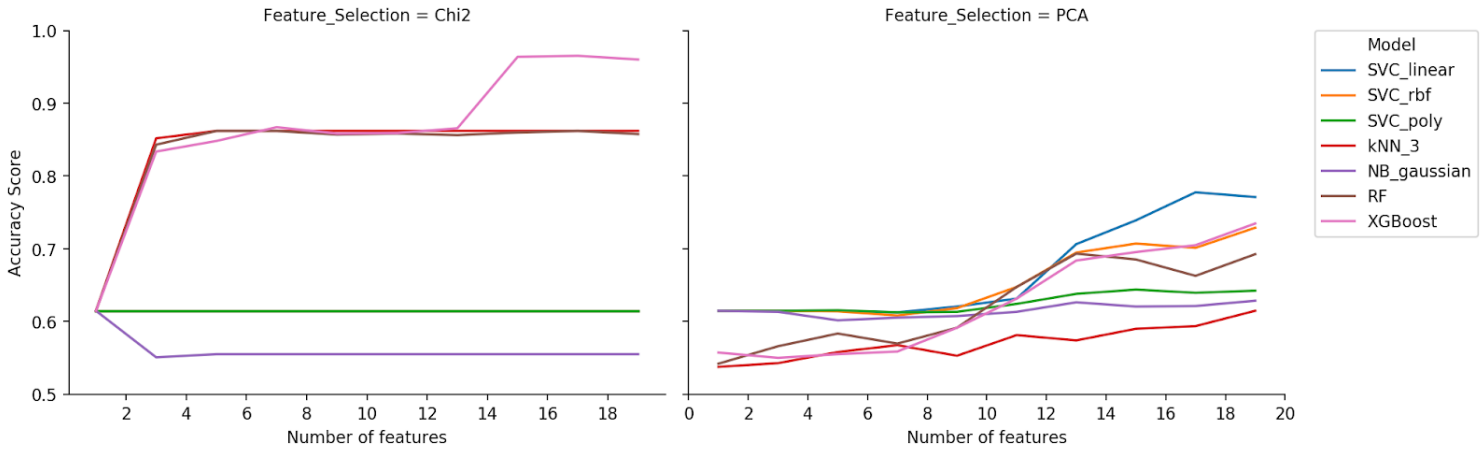

Figure 4.7. Evaluation of Feature Selection Methods for Classification

a results.csv file. This data can then be plotted to visualize how the different models and feature selection techniques performed against each other. An example of a visualization technique can be seen in Figure 4.7 where Chi-Squared and PCA are compared on the machine learning models.

4.6.3 Trend Warning System

The trend warning system also uses the extracted features in the csv file. The warning system interfaces with the user in three ways. By running the warning system the predicted date of the trendline hitting an outlier value for each feature is calculated. For each feature this is stored in a json object that can later be given to the user. The second way the user can interact is exactly that, request the json object. In practice it will be the frontend requesting this information and then parsing it for the user. This list can be filtered by the front end and the user can signify how they want it done. The final way the user can interact with the trend warning system is requesting a visualization of the information. An example of the visualization can be seen in Figure 4.8.

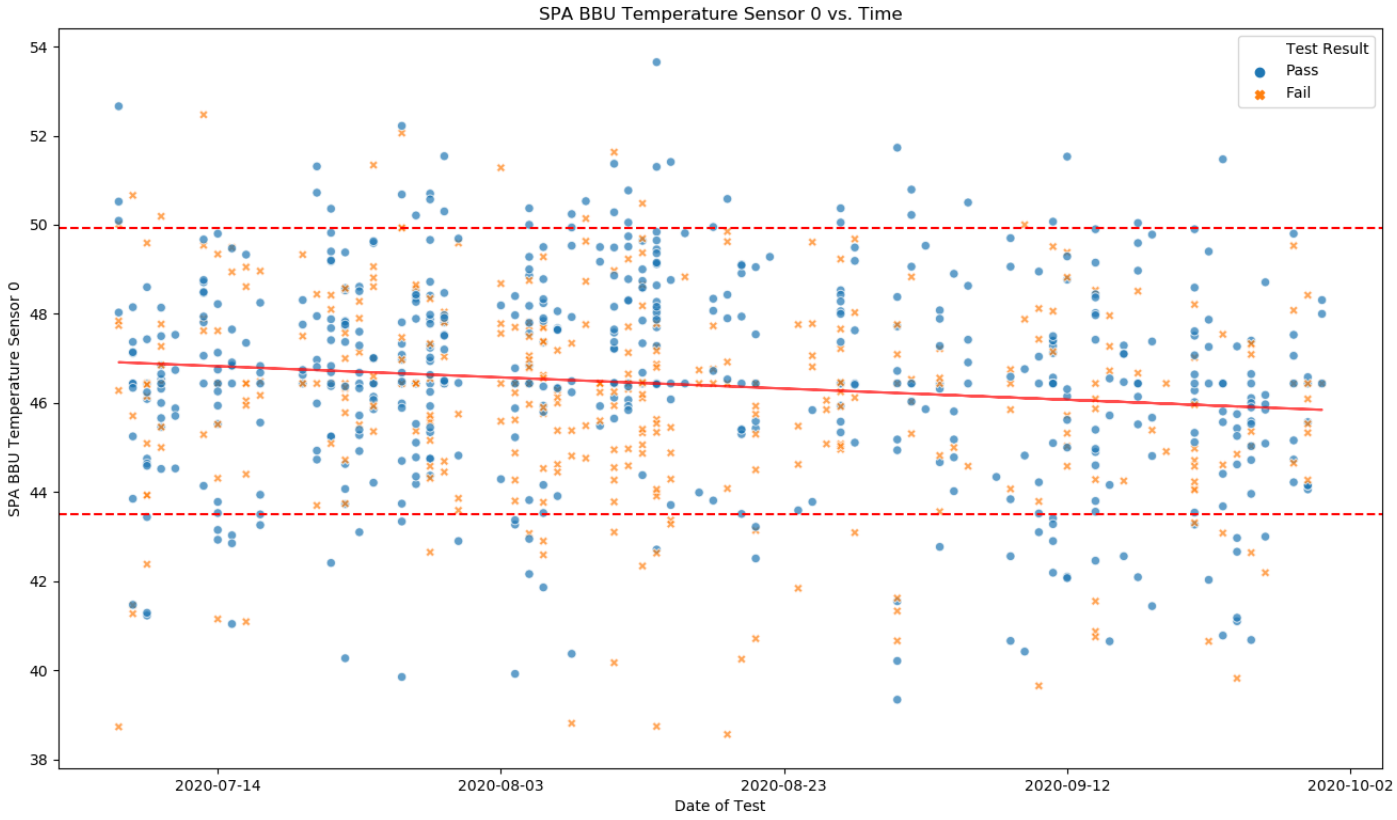

Figure 4.8. Example of Outlier Detection

The image shows the features value over time, the upper and lower bound, and the trend line. This image can be requested by the front end and displayed on the front end so the user can see how a particular feature is trending and if there are many outliers in the data.

5 Results

5.1 Classification

In order to achieve the best results, we trained each classification model using our three feature selection methods: Chi-Squared, Principal Component Analysis (PCA) and Mann-Whitney. Chi-Squared and PCA performed the best and the results from our tests and can be seen below in Figure 5.1. Chi-Squared had better results in general, with the XGBoost algorithm passing 95% accuracy on the test set while PCA's best result was a little below 80% accuracy on the test set.

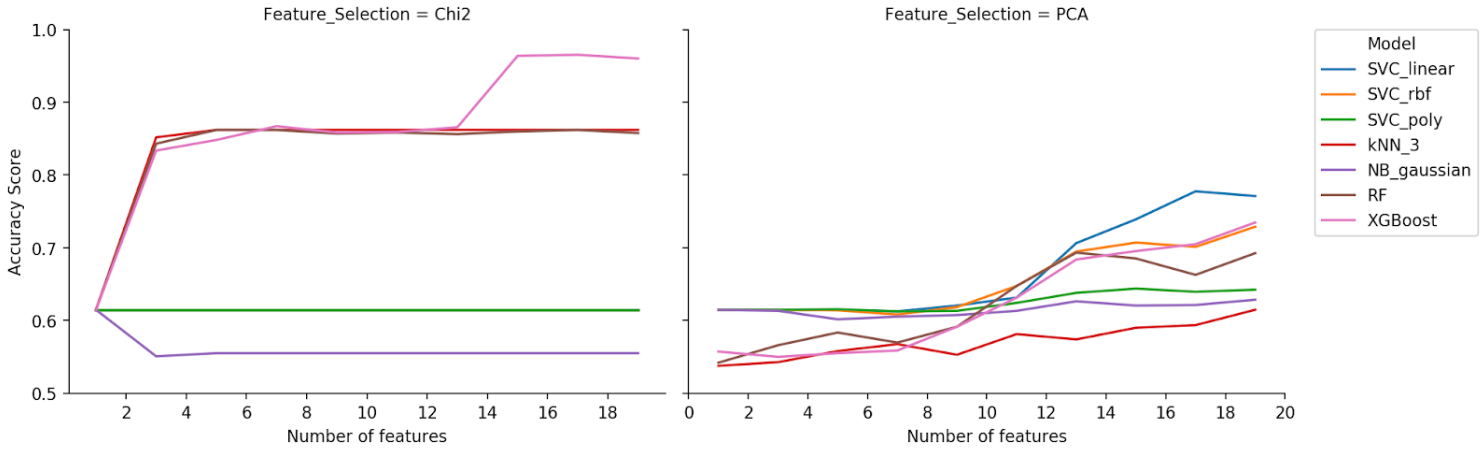

Figure 5.1. Evaluation of Feature Selection Methods for Classification

The data to create these charts can be seen in table 5.1 and table 5.2 below. It should be noted that in the chi squared table, the three SVC models predict that all the data points in the validation set as negative thus giving them a precision and recall of 0.

Model	Number of folds	Accuracy	Accuracy std	Precision	Recall
SVC_{linear}	5	0.6145	0.0268	0	0
SVC_{rbf}	5	0.6145	0.0268	0	0
SVC_{poly}	5	0.6145	0.0268	0	0
kNN_3	5	0.5432	0.0222	0.3990	0.3640
$NB_{gaussian}$	5	0.4414	0.0255	0.3970	0.8622
RF	5	0.5861	0.0410	0.4261	0.2335
XGBoost	5	0.5621	0.0388	0.4123	0.3012

Table 5.1: *Models performance with Chi Squared*

Model	Number of folds	Accuracy	Accuracy std	Precision	Recall
SVC_{linear}	5	0.6516	0.0255	0.6115	0.2938
SVC_{rbf}	5	0.6349	0.0277	0.5575	0.2600
SVC_{poly}	5	0.6058	0.0341	0.45797	0.0666
kNN_3	5	0.5876	0.0287	0.4644	0.4204
$NB_{gaussian}$	5	0.6152	0.0336	0.5145	0.1514
RF	5	0.6378	0.0416	0.5595	0.2944
XGBoost	5	0.6327	0.0295	0.5309	0.4223

Table 5.2: *Models performance with PCA*

Out of all of the machine learning classification algorithms we tried, Gradient Boosting (XGBoost) had the highest accuracy when used with Chi-Squared feature selection. It may not necessarily have the best results on new data presented to it though, since its high accuracy may be the result of overfitting and data snooping. K Nearest Neighbors where K is 3, and Random Forest both had accuracies of around 85% when using Chi-Squared feature selection and these results were the second highest. None of the results mentioned above benefitted much from increasing the number of features at 4 or 6, with the exception of XGBoost. Meanwhile, the results from PCA had steady improvement through the addition of more features. We explore this further in Figure 5.2, where the accuracy with support vector classification steadily rises as more and more features are introduced to its linear and radial basis function (rbf) algorithms. The two go from scoring around just 65% accuracy with 10 features to a little bit over 70% with 40 features, the radial basis function method outperforming the linear one.

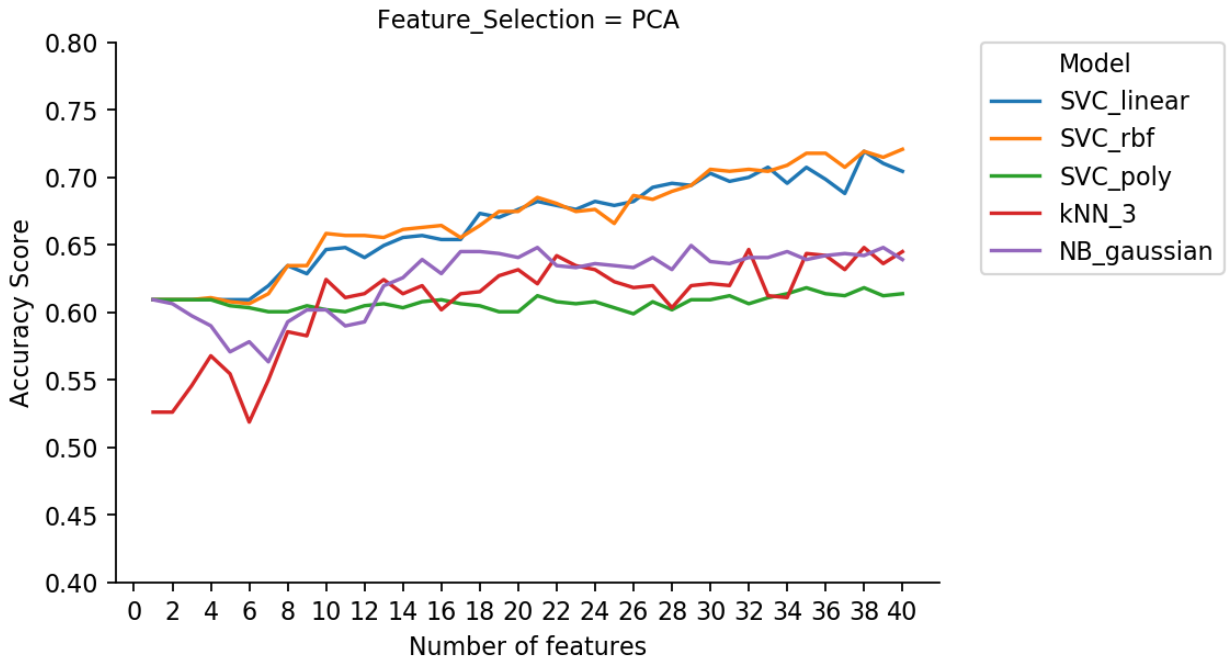

Figure 5.2. Evaluation of algorithms using PCA with up to 40 features

A continual issue we faced were null data values in our dataset. Due to the fact that different testing suites from different companies returned different types of data, some of our features had high null value counts. To combat this we removed features that were more than 80% null values and filled the rest of the null values in with the target mean. To see the full distribution, consult the figure provided in Appendix B. Appendix B is a graph that outlines the features that are more than 10% null values. It represents the number of non null values that passed boards have minus the number of non null values that failed boards have for a particular feature. The idea behind this graph was that it would show us if a null value correlated to a failed board. Unfortunately what this graph revealed to us instead, is that we have more failed boards than passing boards.

5.2 Outlier and Trend Detection

Through our analysis, we found that certain features had distinguishable trends over time. For example, Figure 5.3 depicts the isFailed class of the feature "Temperature_std" as having a lower slope than the passing class as time increases. We see a feature showing a distinguishable trend over time again in Figure 5.4, where the feature "Temperature sensor 0" has a negative slope over time depicted by a solid red line. The dotted red lines, the values of the highest and lowest inliers, represent the range of normal values for the average to be within. Once the solid red line crosses out of this territory it becomes worrisome and should be investigated. Using regression we predict the date when this event will occur and warn Dell once that date is within a certain amount of time.

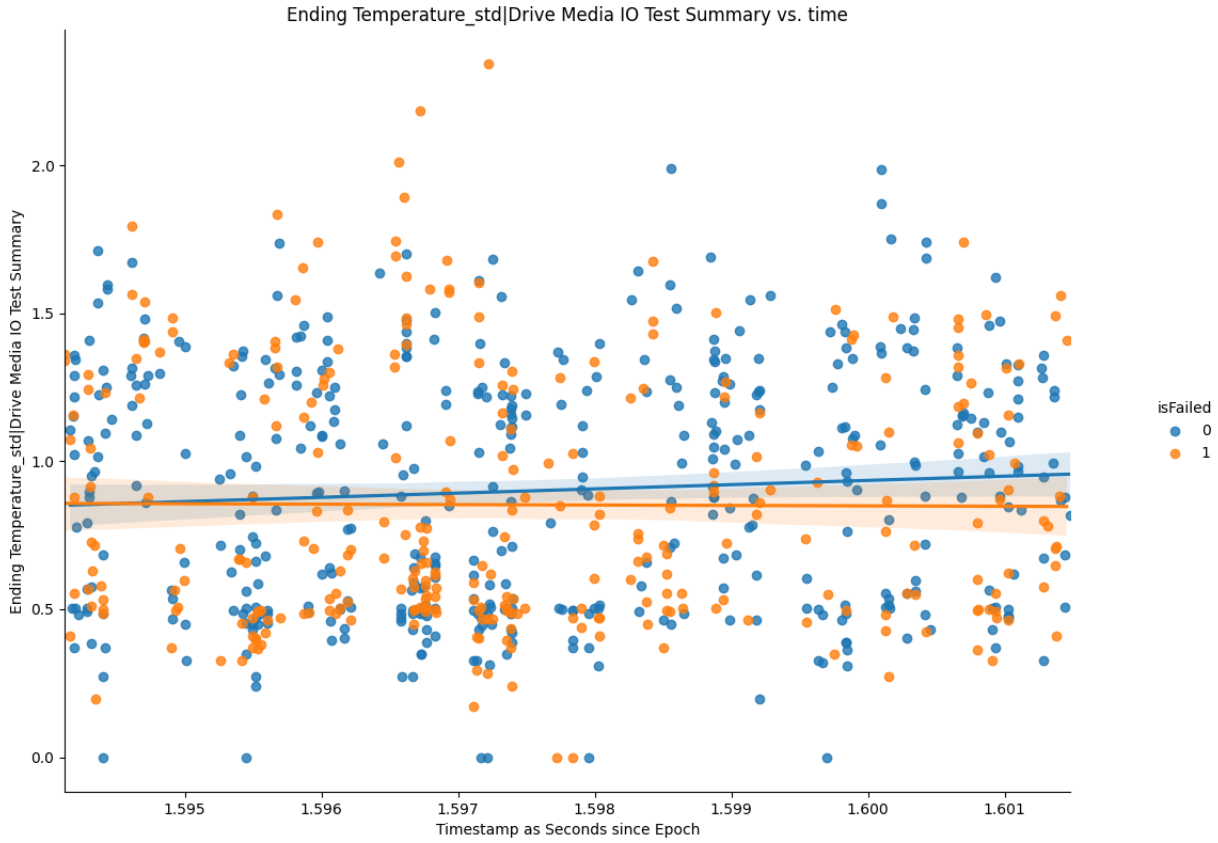

Figure 5.3. Example of distinguishable trends over time

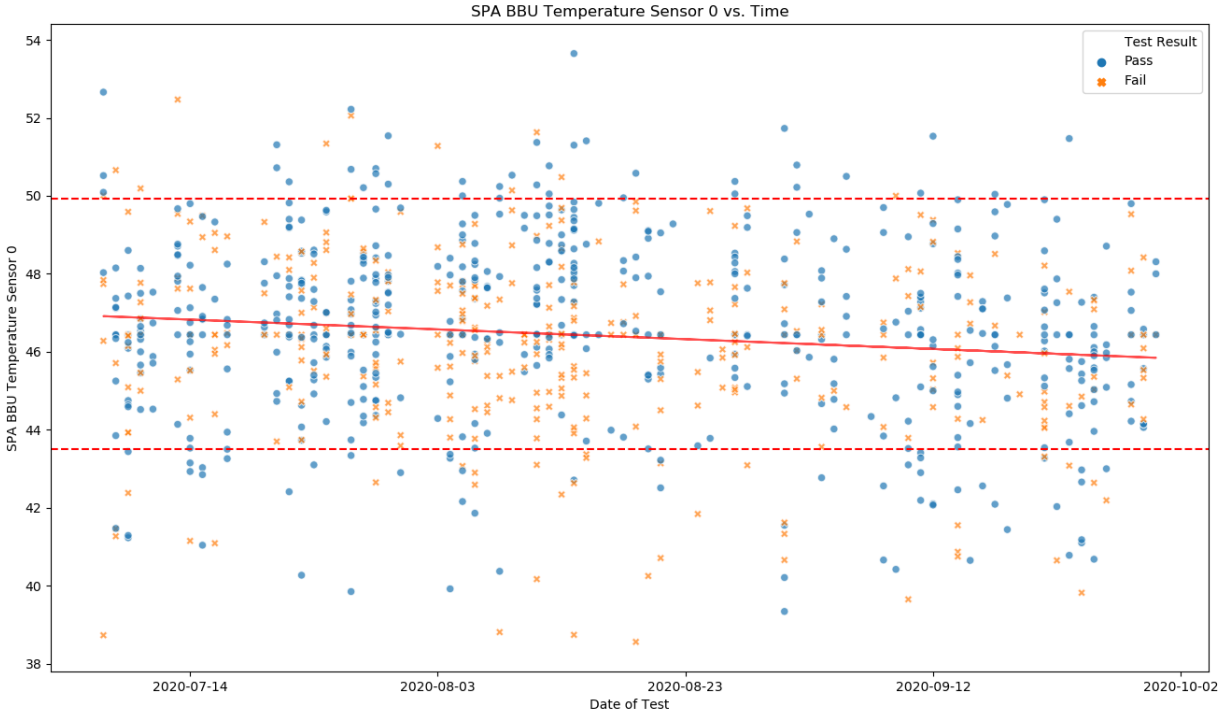

Figure 5.4. Example of outlier and trend detection

6 Conclusions and Deliverables

6.1 Standardized Log Data Format

When the Warnado boards go through the testing suite, they run through multiple tests. After each test, results are written out to various files such as UutItems.json. This file contains a list of each test run and the time stamp it started and ended at. This was utilized to create the feature that describes how long each test step took to run. Another file that was focused on was UutDataLog.json. This was the largest file generated by the test and the one that the most time was spent on. The rows in the log files are as follows: RowNumber, LogEntryId, ParentRegionId, PlaceholderRegionId, EntryTime, EntryLevel, SourceType, SourceName, Tag, MessageFormat, MessageData, and HasErrors. The three columns we focus on were SourceName, Tag, MessageData, and HasErrors. SourceName has the name of the test step. The Tag column describes what is happening in the current line of data. This can be the start or end of a test step or logging the results. We utilized this column to split the data into different test steps. The MessageData column contains the data that was output from the test. This column is where we pulled most of our features from as it has distinct values. The format of this column was not consistent and ranged from json tables to human readable text. Finally the HasErrors column was a boolean flag that was true if a test had errors and false otherwise. This column splits the data into pass and fail as laid out by MIL.

As mentioned above, there is no standardization for the MessageData column. Each test step has its own format to output the results of the test. This made it very difficult to extract features from the tests as for each test we had to write custom code to extract the output. The format of the output was very human readable, but was unstructured and was difficult to parse with a computer as shown in Figure 6.1.

```
Debug
  PlatypusDatas
    ppdata
      timestamp: 2020-09-25 04:03:38.334757
      devices
        device
          name: CDI
          guid: 0
          type: Root
          help_hash: 0c41fb6b76203d19fe118e4493978a19c7bc2c93
          position
            dynamic: 0_A
            descriptive
              enclosure: 0
              sp: A
          devices
            device
              name: Warnado
              guid: d0415348-fed0-11ea-b9d9-006016b23f8c
              type: Platform
              help_hash: 5652de104c82c83deb1c808fb0d5742b15f7d795
              position
                dynamic: 0_A
                descriptive
                  enclosure: 0
                  sp: A
```

Figure 6.1. Human Readable output

Based on our findings, we compiled a list of test steps that should be reformatted into JSON formatting. In addition, we wrote a document explaining our log data standardization proposal, best practices for JSON formatting, and an itemized list of test steps with a brief description of their current state. This proposal can be viewed in full in Appendix C.

We attempted to write a generalized text reader, but this still had to be custom fit to each test step we wanted to extract features from. Of the 149 test steps throughout the data, we were only able to extract features from 6 of them. This significantly hindered the development of this project not only by the lack of features we had access to, but also the time it took to write custom feature extraction for the test steps we used. While writing custom extraction code, we did find an output that was in JSON and in the format of a table. Writing code to extract these features was much simpler than parsing through than the other outputs we had come across. Our recommendation for Dell EMC is to standardize the output of each test step to match this format to allow for future data scientists to be able to more easily extract features and create models.

The format of the JSON is very similar to that of a table. There are three keys to the object: Title, Columns, and Rows. The title key will give the name of what test was run and give context to the data currently being displayed. This will allow each feature to have a deeper description than just what test step it took place in, but also what exactly was being tested. The columns key will be an array describing the columns in the table. These column names will become individual features for the particular test. If the table contains more than one value each column can be split into four different descriptors: mean, standard deviation, maximum, and minimum of column. These descriptors can become individual features. This implementation can be utilized by future data scientists as part of feature engineering and we encourage Dell EMC to just keep the list of values and not the descriptors. The final key, rows, will be an array of arrays. The first layer of arrays (the array of arrays) will be a list of rows of data, where each row of data is another array. Each row of data will be an array where each index aligns with the column definition from above. An example of how the output data should be formatted can be seen in Figure 6.2.

{"Title":"Initial Drive Expander Phy count Summary",

"Columns":["Drive
Pos","invalid_dword_count","disparity_error_count","code_violation_error_count","loss_of_dword_sync_co
unt","phy_reset_failed_count","phy_change_count","crc_pmon_accumulation_count","in_connection_crc_er
ror_count"],"

“Rows":[["0_A_2_0_2_0_5","7","7","7","0","0","11","0","0"],["0_A_2_0_2_0_6","62","63","46","2","0","11","0","0"],["
0_A_2_0_2_0_7","72","74","74","2","0","11","0","0"],["0_A_2_0_2_0_0","190","194","136","4","0","11","0","0"],["0_
A_2_0_2_0_1","179","184","184","5","0","11","0","0"] ...

Figure 6.2. Proposed Output for each Test

6.2 Warning for Trend Detection

Based on the trends we found through outlier and trend detection, we created a program that would alert and flag Dell EMC of specific features that would eventually drift into the outlier range of values. We referenced these features as having reached “criticality”.

6.2.1 Front End

The front end of the program was written using JavaScript’s React library. The design was kept simplistic, and covers just two pages, as there is only one main function. The main page (shown in Figure 6.3) displays to the user the number of logs expected to reach a critical state within the next month. This number is generated from the last time the program was run. To run the program, the user must click the first of the two main buttons “Run Warning System”. This will run on the ZIP file indicated by a parsed argument.

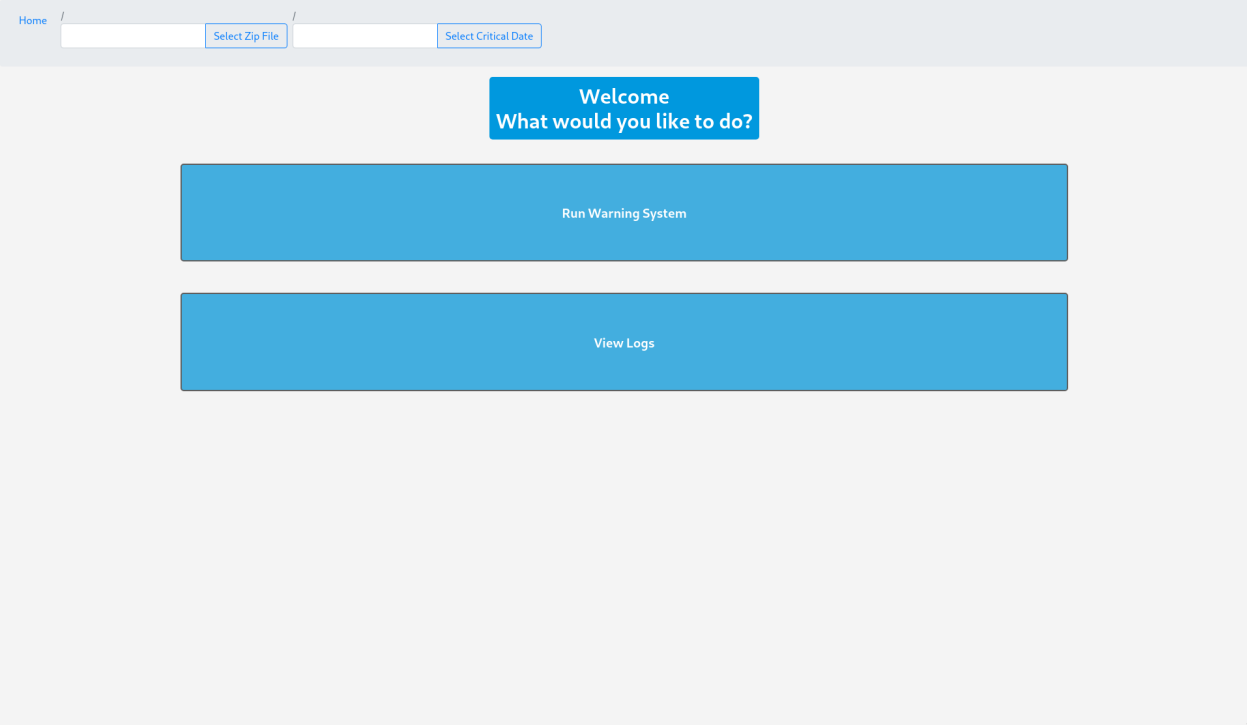

Figure 6.3. A screenshot of the first page of the warning system UI

The second button, “View Logs”, links to the next page, shown in Figure 6.4. On entering this page, the user is shown the most critical log. At the top, there is a header which states that the “[Feature Name] will reach a critical value of [Outlier Value] by [MM/DD/YYYY]”. Alongside this header, the original test log file and test step that this feature was extracted from are listed. Underneath the information, a time series plot is presented with the regression line and outlier range marked. On the left side of the page, there is a scroll bar. The bar contains each of the extracted features. By default, this is sorted from nearest to furthest date of criticality. The bar can also be sorted by individual test steps for further investigation between similar features. When clicking on a feature within the bar, the information displayed on the page will change to that of the selected feature.

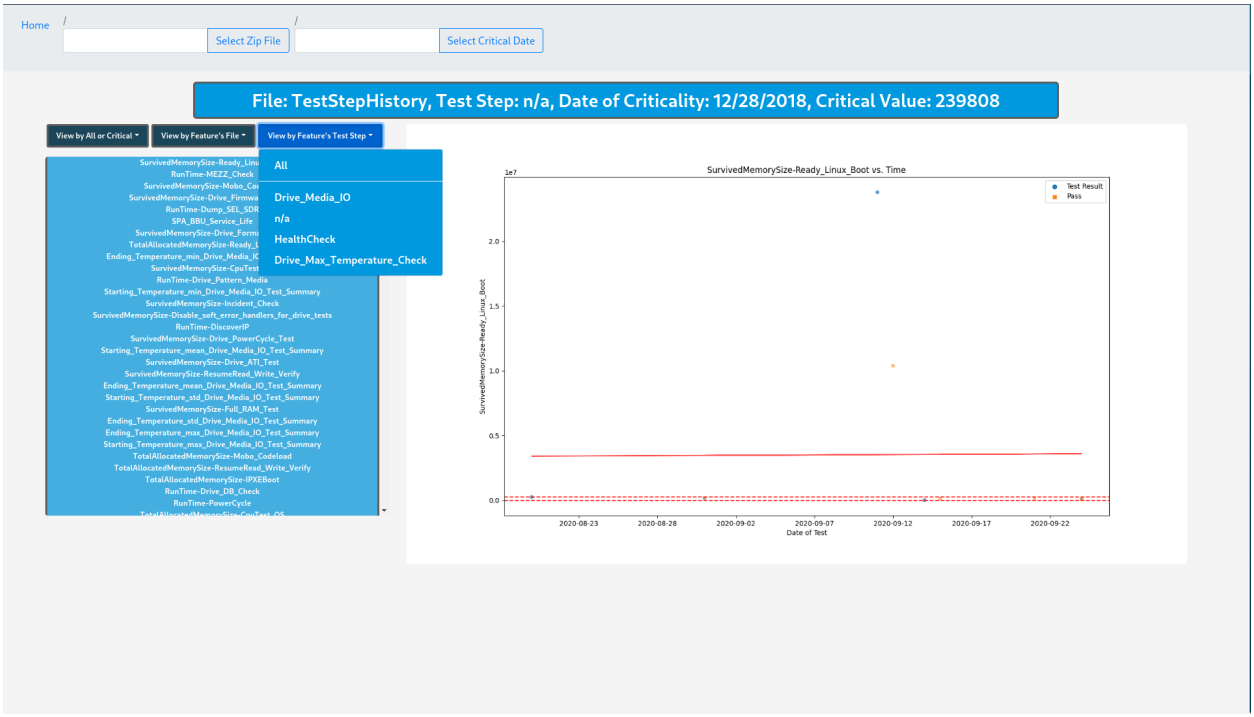

Figure 6.4. A screenshot of the second page of the warning system UI

6.2.2 Back End

The back end contains three endpoints. The first endpoint will execute the main warning system script when the "Run Warning System" button is pressed. The second endpoint retrieves a list of features, as a JSON file, generated by the main system script. Lastly, the third endpoint will generate and retrieve individual charts each time a different feature is selected from the scroll bar. This will save time by preventing the need for every chart to generate at once when the system is run.

The main warning system script was written in Python. This script is a portion of the main pipeline mentioned in section 4.6.1. The pipeline is called using specific arguments to indicate that a zip file of raw log data must be formatted, feature extraction must be run, and the warning system script must be called. The warning system script first collects timestamps for each unique motherboard, these will be used when plotting the time series. One feature at a time, the script processes an array of feature values (one for each motherboard). The Isolation Forest classifier is

$$X = \frac{(y-b)}{m}$$

Figure 6.5. Linear Equation solved for Outlier Date

fit on these features, and each value is marked as an inlier or an outlier. SciPy's statistical package is used to calculate the linear regression function for the data. This gives us the slope and y-intercept. By inputting the maximum and minimum inlier values sequentially as y, x is solved for. As this is calculated for both the maximum and minimum inlier values, two x-values are generated. The greater x-value is taken as the date of criticality because the smaller x-value would represent a date in the past.

The script also contains functionality used to generate time series charts for each feature. This function takes in the before mentioned data processing results, and uses them to create a time series scatter plot using MatPlotLib and Seaborn.

Lastly, the script saves a JSON file containing all needed information to be displayed in the front end. Following are the four attributes collected for each feature: the date of criticality, the feature value at this date, and both the file and the test step from which the feature was extracted.

6.3 Final Conclusions

Through this project, we were able to identify some key factors that would improve Dell EMC's workflow. Determining how log data can be improved as well as developing a warning system and application for trend detection will serve to improve Dell EMC's design and productivity. Standardizing the log format readies Dell EMC for the future of big data and AI by having structured data that can easily be fed into a machine learning model. This will save future data scientists weeks of data cleaning and feature extraction to allow the process to be expedited. The warning system enables Dell EMC engineers to easily monitor motherboard health trends. This exploratory project gave the Dell EMC team a better understanding of their motherboard log data, and will serve to increase the efficiency of future projects.

7 References

Castiglioni, P. (2005). Reverse Arrangement Test. In Encyclopedia of Biostatistics (eds P. Armitage and T. Colton). DOI:10.1002/0470011815.b2a12059

Chigurupati, A., Thibaux, R., Lassar, N. (2016). Predicting hardware failure using machine learning. Tucson, AZ: Annual Reliability and Maintainability Symposium (RAMS). DOI: 10.1109/RAMS.2016.7448033

Doran, G., Ray, S. (2014). A theoretical and empirical analysis of support vector machine methods for multiple-instance classification. Mach Learn 97. Retrieved from https://doi-org.ezpxy-web-p-u01.wpi.edu/10.1007/s10994-013-5429-5

Du, M., Li, F., Zheng, G., Srikumar, V. (2017). DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. New York, NY, USA: Association for Computing Machinery. DOI:https://doi.org/10.1145/3133956.3134015

Fulp, E., Fink, G., Haack, J. (2008). Predicting Computer System Failures Using Support Vector Machines. Retrieved from https://static.usenix.org/events/wasl08/tech/full_papers/fulp/fulp.pdf

Gao, R., Zhang, T., Sun1, S., Liu, Z. (2019). Research and Improvement of Isolation Forest in Detection of Local Anomaly Points. Journal of Physics: Conference Series. doi:doi:10.1088/1742-6596/1237/5/052023

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2013). An introduction to statistical learning : with applications in R. New York :Springer

Gonfalonieri, A. (2019, November 07). How to Implement Machine Learning For Predictive Maintenance. Retrieved September 14, 2020, from https://towardsdatascience.com/how-to-implement-machine-learning-for-predictive-maintenance-4633cdbe4860

H.B. Mann, D.R. Whitney (1947), "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other," The Annals of Mathematical Statistics, vol. 18, no. 1, pp. 50-60.

Jin X., Xu A., Bie R., Guo P. (2006) Machine Learning Techniques and Chi-Square Feature Selection for Cancer Classification Using SAGE Gene Expression Profiles. In: Li J., Yang Q., Tan AH. (eds) Data Mining for Biomedical Applications. BioDM 2006. Lecture Notes in Computer Science, vol 3916. Springer, Berlin, Heidelberg. Retrieved from https://doi.org/10.1007/11691730_11

Liu, F. T., Ting, K. M. Zhou, Z. (2008). Isolation Forest. Italy: 2008 Eighth IEEE International Conference on Data Mining. DOI: 10.1109/ICDM.2008.17.

Melki, G., Cano, A., Ventura, S. (2018). MIRSVM: Multi-instance support vector machine with bag representatives. Retrieved from https://doi.org/10.1016/j.patcog.2018.02.007.

Murray, J. F., Hughes, G. F., Kreutz-Delgado, K. (2005). Machine Learning Methods for Predicting Failures in Hard Drives: A Multiple-Instance Application. Retrieved September 14,

2020, from https://jmlr.csail.mit.edu/papers/volume6/murray05a/murray05a.pdf

National Institute of Standards and Technology. (N. D). Introduction to Time Series Analysis.Gaithersburg, MD: National Institute of Standards and Technology. Retrieved from www.itl.nist.gov/div898/handbook/pmc/section4/pmc4.htm.

Poursaeidi, M.H., Kundakcioglu, O.E. Robust support vector machines for multiple instance learning.Ann Oper Res 216, 205–227 (2014). Retrieved from https://doi-org.ezpxy-web-p-u01.wpi.edu/10.1007/s10479-012-1241-z

Scikit-Learn. (2020). 2.7. Novelty and Outlier Detection. Scikit-Learn. Retrieved from scikit-learn.org/stable/modules/outlier_detection.html.

8 Appendices

8.1 Appendix A: Feature Variance in Data Exploration

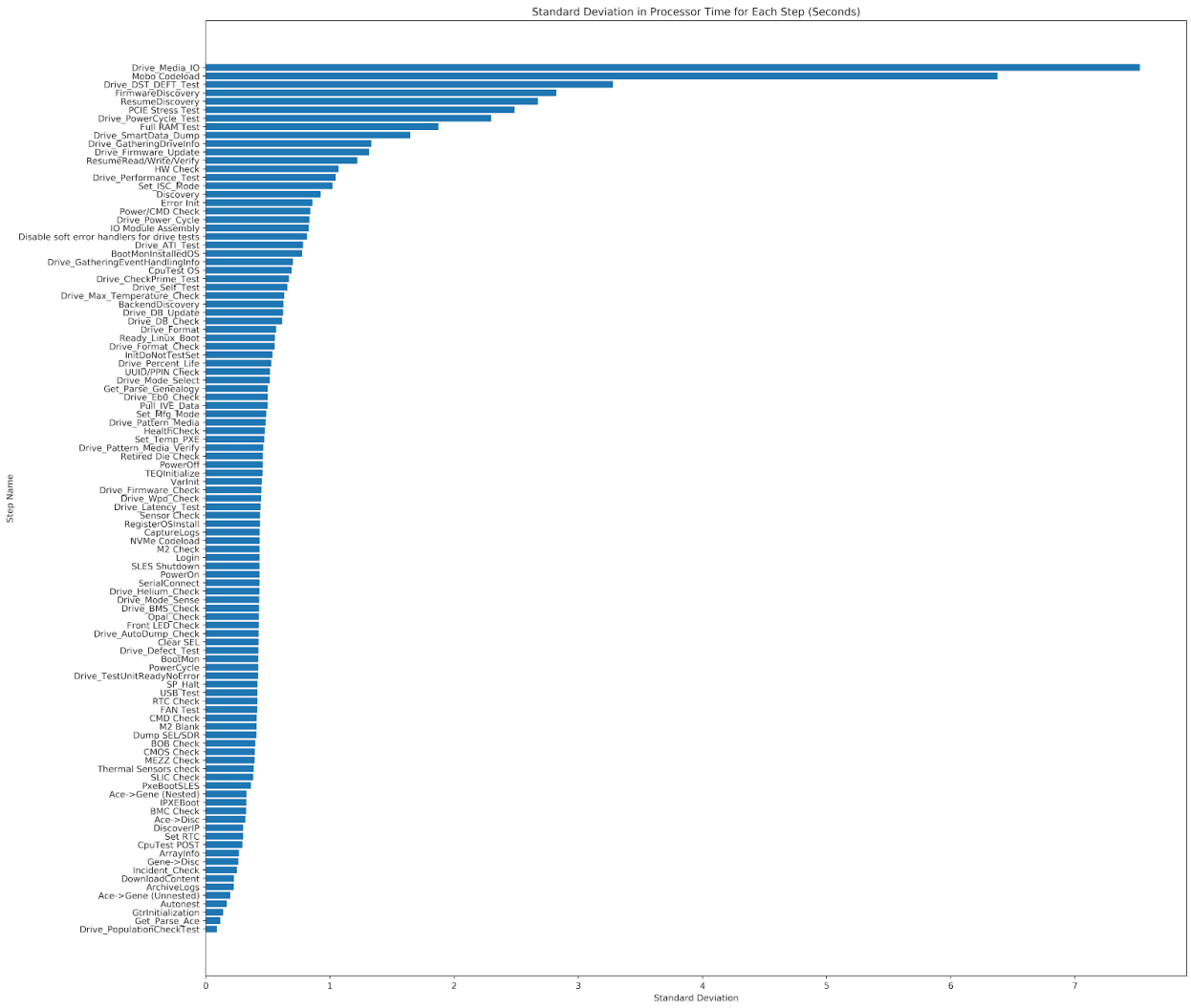

Figure 8.1. Std. Deviation of Processing Times by Step

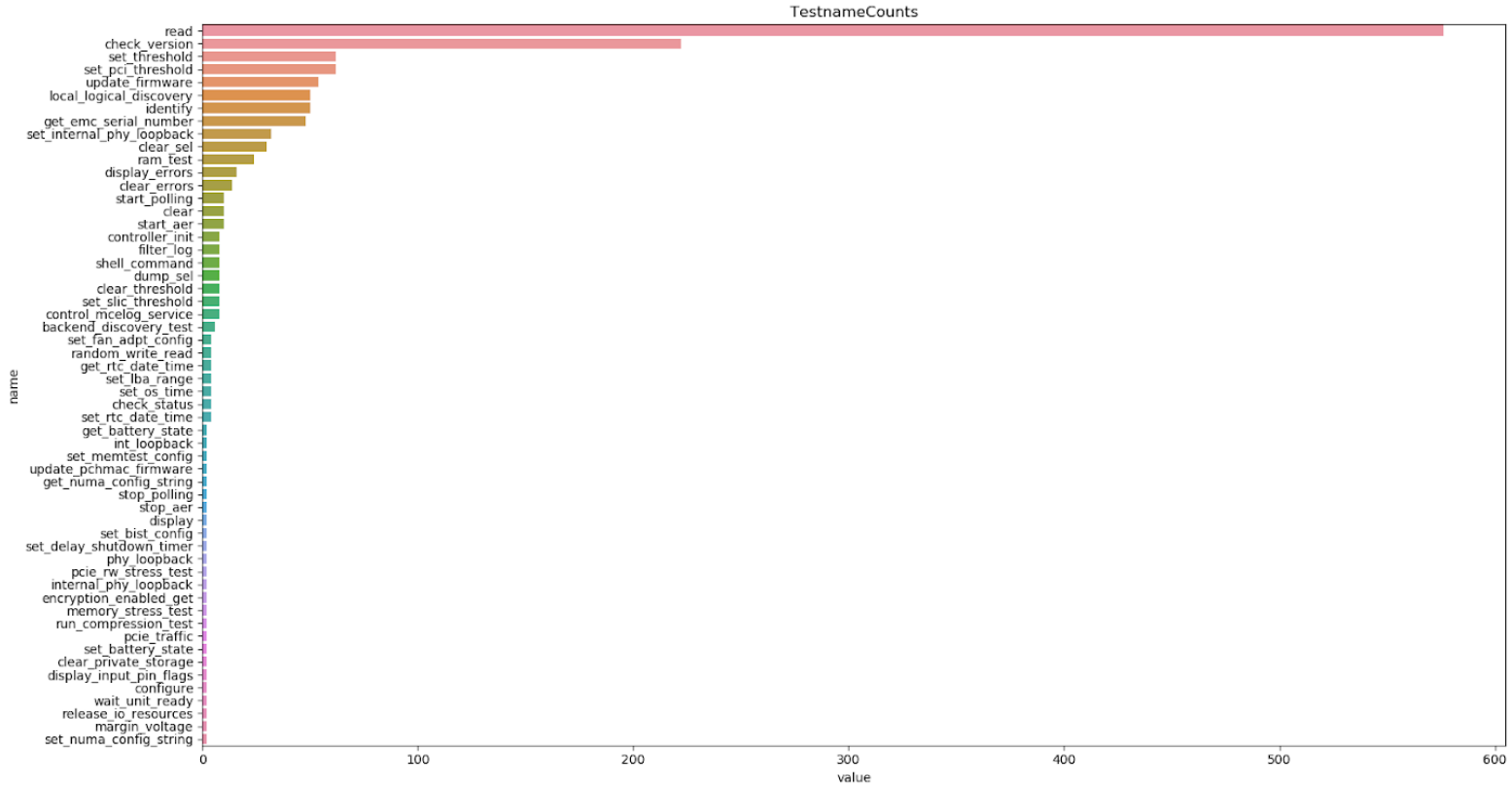

Figure 8.2. Counts of Testnames across Logs

8.2 Appendix B: Nullcounts

Appendix B is a graph (following page) that outlines the features that are more than 10% null values. It represents the number of non null values that passed boards have minus the number of non null values that failed boards have for a particular feature.

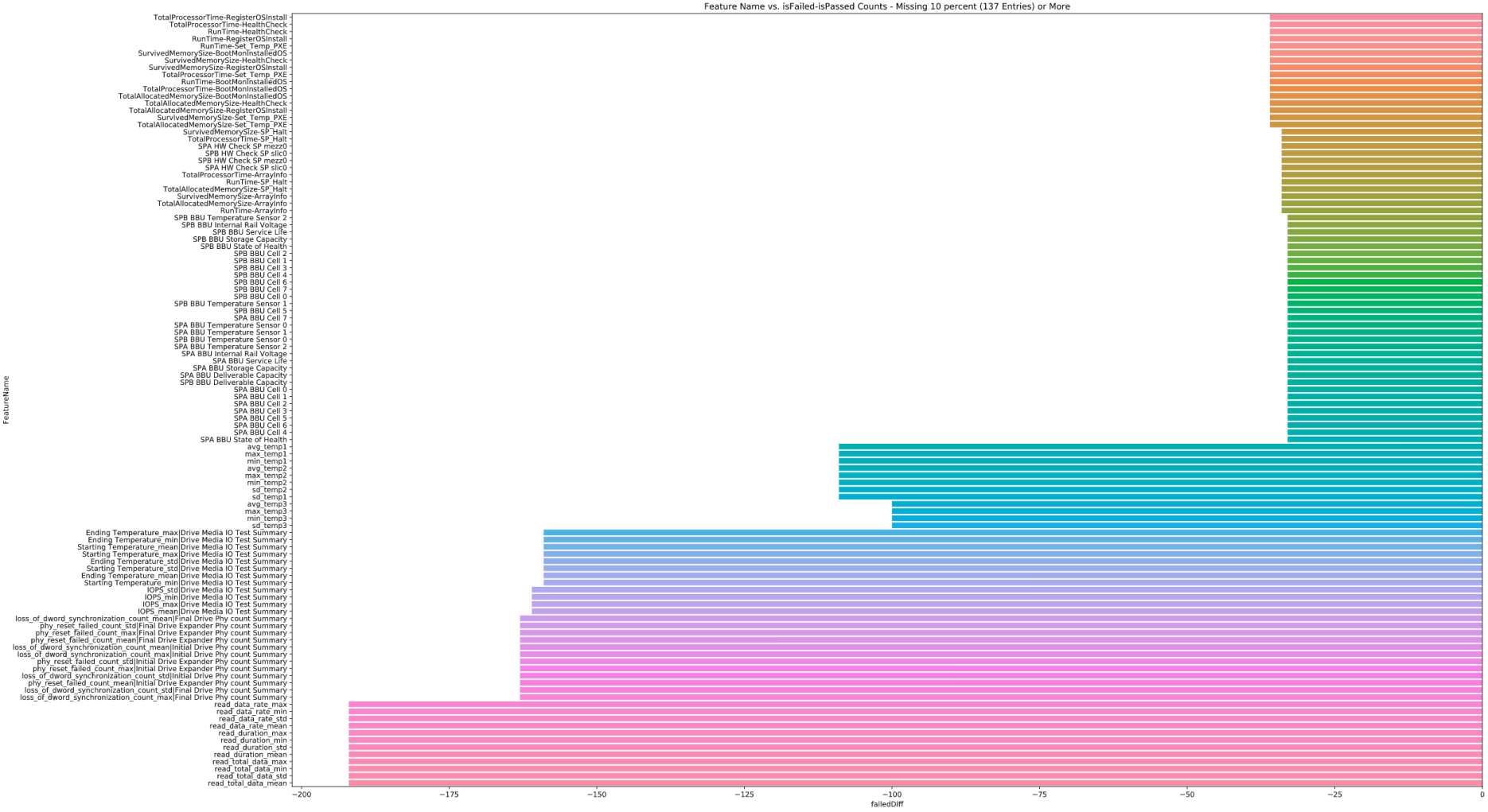

Figure 8.3. Null counts

8.3 Appendix C: Log Standardization

See the following six pages for the Warnado Log Data Standardization Deliverable.

Warnado Log Data Standardization

Proposed by the 2020-2021 WPI MQP team.

Recommended Format:

Through processing of the Warnado motherboard test logs, we found variance in data structures between test steps. We have determined an ideal format for the data, which would improve the efficiency of future projects.

We recommend a JSON format, which is a language-independent format consisting of attribute-value pairs.

Why this Format is Valuable:

Firstly, it is important to note that any consistent data format is preferable to inconsistency. This enables data scientists to create standardized feature extraction or other processing scripts, which will work on all current and future log files. This can save a tremendous amount of time and infinitely increase scalability of the usage of the log data.

We determined JSON as the most ideal format for two further reasons. One, it is both human and computer readable. The attribute-value pairs can be easily discerned by the human eye. More importantly, these pairs are computer readable, and can be parsed by many languages. Most IDEs can format JSON, and there are JSON specific web and software viewers. Second, the table-like format of JSON allows for details to be included with the log data. In the example shown in **Figure 1**, the data contains three attributes: ‘Title’, ‘Columns’, and ‘Rows’. The title allows for context to be provided for this portion of the test step. The column names are descriptors that will become individual feature names for the test. The rows contain the actual data, which can be featurized using statistical summaries such as the mean.

{"Title":"Initial Drive Expander Phy count Summary",

"Columns":["Drive Pos","invalid_dword_count","disparity_error_count","code_violation_error_count","loss_of_dword_sync_count","phy_reset_failed_count","phy_change_count","crc_pmon_accumulation_count","in_connection_crc_error_count"],

“Rows":[["0_A_2_0_2_0_5","7","7","7","0","0","11","0","0"],["0_A_2_0_2_0_6","62","63","46","2","0","11","0","0"],["0_A_2_0_2_0_7","72","74","74","2","0","11","0","0"],["0_A_2_0_2_0_0","190","194","136","4","0","11","0","0"],["0_A_2_0_2_0_1","179","184","184","5","0","11","0","0"] ...

Figure 1. A Good Example Within the Log Data from the driveMediaIO Test Step

The example in **Figure 2** contains valuable information but requires manual attention to be parsed. As this is not a standard format, we had to review the structure of this data in order to create a parsing script. It is inefficient to manually create different scripts for each test step, and it can be time consuming to ensure we are collecting all valuable information. Most of the test steps contained within the UutDataLog.json 'MessageData' attribute, were in varying formats other than JSON. This inconsistency costs time which would be better spent analyzing the parsed information.

2:|EMPTY|\r\nSPA IO Module Mezz 0 Port 3:/|EMPTY|\r\nSPA BBU Temperature Sensor 0:/40.30C|\r\nSPA BBU Temperature Sensor 1:|31.72C|\r\nSPA BBU Temperature Sensor 2:|31.51C|\r\nSPA BBU Internal Rail Voltage:|12.28V|\r\nSPA BBU Service Life:|1days|\r\nSPA BBU Storage Capacity:|22.80WHr|\r\nSPA BBU Deliverable Capacity:|22.80WHr|\r\nSPA BBU State of Health:|97%|\r\nSPA BBU Cell 0:|3.65V|\r\nSPA BBU Cell 1:|3.64V|\r\nSPA BBU Cell 2:|3.72V|\r\nSPA BBU Cell 3:|3.57V|\r\nSPA BBU Cell 4:|3.64V|\r\nSPA BBU Cell 5:|3.65V|\r\nSPA BBU Cell 6:|3.65V|\r\nSPA

Figure 2. A Bad Example Within the Log Data from the healthCheck Test Step

JSON Formatting Guidelines:

When formatting data in JSON notation, there are several conventions that should be followed.

Keys:

- Names should be as clear and explicit as possible to enhance readability
- The plurality of the key should match the key's associated value
    - Single values should have a singular key name while an array of values should have a plural key name

```
{ "temp" : 46 }

{ "temps" : [46, 21, 101, 20] }
```

Figure 3. Example of key plurality matching

Data Structure:

- Data should be 'flattened' when possible
    - Flattening is the practice of only utilizing Key-Value pairs, rather than nested data
    - Do not group and nest data unnecessarily
    - Data should only be nested when it makes semantic sense

```
{ "SensorValues" : {
        { "temperatureSensor" : 46 },
        { "voltageSensor" : 1.27 }
```

```
        }
}
```

Figure 4. Example of Unnecessary Data Grouping

```
{ “temperatureSensor” : 46,
  “voltageSensor” : 1.27 }
```

Figure 5. Example of flattening data

Data Integrity:

- Keep types consistent in value arrays and lists
- Omit null and ‘none’ values from the dataset

For further information, we recommend consulting the Google JSON Style Guide.

Priority List of Test Steps to Re-format:

1. Formats which required regex parsing of the message data
    a. Hw_check
    b. health_check
    c. slic_check
2. Format that required parsing for ‘Summary’ in ‘MessageData’
    a. Drive_media_io

Test Steps with good formatting:

JSON Output
Ace->Disc
Ace->Gene (Nested)
Ace->Gene
Autonest
Discovery
Drive_ATI_Test
Drive_AutoDump_Check
Drive_BMS_Check
Drive_CheckPrime_Test
Drive_DB_Check
Drive_DB_Update
Drive_Defect_Test
Drive_DST_DEFT_Test
Drive_Eb0_Check

Drive_Firmware_Check
Drive_Format
Drive_Format_Check
Drive_GatheringEventHandlingInfo
Drive_Helium_Check
Drive_Latency_Test
Drive_Max_Temperature_Check
Drive_Media_IO
Drive_Mode_Select
Drive_Mode_Sense
Drive_Pattern_Media
Drive_Pattern_Media_Verify
Drive_Percent_Life
Drive_Performance_Test
Drive_PopulationCheckTest
Drive_PowerCycle_Test
Drive_Power_Cycle
Drive_Self_Test
Drive_SmartData_Dump
Drive_TestUnitReadyNoError
Drive_Wpd_Check
Gene->Disc
Opal_Check
Pull_IVE_Data
Retired Die Check

Test Steps in need of reformatting:

Tables

Clear SEL - Prints out table with | as seperator
Dump SEL/SDR - Prints out table with | as seperator
Sensor Check - Prints out table with | as seperator
Drive_GatheringDriveInfo - Key Value list separated by :
Drive_Firmware_Update - Key values list separated by :
SLIC Check - Prints human readable list with :
M2 Check - Table with no separators

Console Results with ':' delimiters

BOB Check
CMD Check
CMOS Check
HealthCheck
Thermal Sensors check

Console Results with '=' delimiters

BMC Check
FAN Test
MEZZ Check
RTC Check
TEQInitialize
USB Test

Text Output:

BootMon
CpuTest POST
DiscoverIP
PowerCycle
PowerOff
PxeBootSLES
SerialConnect
Set_ISC_Mode
Set RTC
UUID/PPIN Check

Just Console Result:

ArchiveLogs
ArrayInfo
BootMonInstalledOS
CaptureLogs
DownloadContent
Front LED Check
Incident_Check
IPXEBoot
HW Check
Mobo Codeload
M2 Blank
NVMe Codeload
PowerOn
Ready_Linux_Boot
RegisterOSInstall
SP_Halt
Set_Mfg_Mode
Set_Temp_PXE
SLES Shutdown

Prints human readable lists with limited information

BackendDiscovery
CpuTest OS
Disable soft error handlers for drive tests
Error Init
FirmwareDiscovery
Full RAM Test
IO Module Assembly
PCIE Stress Test
Power/CMD Check
ResumeDiscovery