Brigham Young University
BYU ScholarsArchive

All Theses and Dissertations

2016-12-01

Recognition of Infrastructure Events Using Principal Component Analysis

Lane David Broadbent
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

 Part of the Systems Engineering Commons

BYU ScholarsArchive Citation

Broadbent, Lane David, "Recognition of Infrastructure Events Using Principal Component Analysis" (2016). *All Theses and Dissertations*. 6197.
https://scholarsarchive.byu.edu/etd/6197

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Recognition of Infrastructure Events Using Principal Component Analysis

Lane David Broadbent

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Joseph J. Ekstrom, Chair
Dale C. Rowe
Kevin B. Tew

School of Technology

Brigham Young University

Copyright © 2016 Lane David Broadbent

All Rights Reserved

ABSTRACT

Recognition of Infrastructure Events Using Principal Component Analysis

Lane David Broadbent
School of Technology, BYU
Master of Science

Information Technology systems generate system log messages to allow for the monitoring of the system. In increasingly large and complex systems the volume of log data can overwhelm the analysts tasked with monitoring these systems. A system was developed that utilizes Principal Component Analysis to assist the analyst in the characterization of system health and events. Once trained, the system was able to accurately identify a state of heavy load on a device with a low false positive rate. The system was also able to accurately identify an error condition when trained on a single event. The method employed is able to assist in the real time monitoring of large complex systems, increasing the efficiency of trained analysts.

Keywords: syslog, log analysis, principal component analysis

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor, Dr. Ekstrom, as well as my committee members for their suggestions and guidance.

I would also like to thank my family for their patience and support.

Finally, I must express my profound gratitude to my wife, Holly, for providing me with unfailing support, optimism, and encouragement throughout the process of researching and writing this thesis. I could not have completed it without her. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Background

The ability to monitor is a crucial aspect in the operation of any IT system. This is often done using system logs, hereafter referred to as syslog messages, which can be collected at significant scale. The ability for a small number of engineers and analysts to process what is an ever-increasing volume of log data has benefits for the IT community.

The proliferation of cloud services, the Internet of Things, and an increasing reliance on information technology has resulted in massive interconnected systems. Theses systems generate a significant amount of log data intended to assist in the operation and troubleshooting of the system. The log management system utilized by Brigham Young University's Office of Information Technology (BYU OIT), for example, collects millions of log messages a day. The volume of log data can be overwhelming for those tasked with its analysis and monitoring, allowing critical information to be overlooked. The information collected through logging is often archived and used for eventual failure analysis instead of allowing for active monitoring and understanding of these complex IT environments.

Logging is often performed using messages following the Syslog (Gerhards, 2009) standard, standardizing the format and transmission of log messages. The Syslog message format requires that pertinent information about the origin of the message, such as messages severity,

originating process, and host, be present in all messages. This allows for information to be readily extracted and interpreted by an automated process, aiding in monitoring.

Active monitoring, particularly being able to detect crucial events and determine the state of a system in a timely fashion, is the problem this research seeks to solve. An analyst needs to be able to quickly analyze a large volume of data. Experience shows that log analysis can often be a tedious process that requires a significant amount of time and experience in both the system being monitored and the log management system in use. While performing log analysis and monitoring, an analyst can be faced with a tough choice: limit their scope to a small number of systems to be monitored in detail or monitor a large number of systems for significant events -- often after an event has occurred. The effectiveness of an analyst can be greatly improved if a means of automated analysis can be used.

A significant amount of the value of an analyst comes from his experience, or being an "expert" in the systems being monitored. This experience and intuition is difficult to transfer to an automated process. However, there exist methods of statistical analysis that allow for insight without the necessity of expertise in a system. One such method is that of Principal Component Analysis. Principal Component Analysis, or PCA, has long been used to simplify and gain insight into multidimensional data.

Principal Component Analysis (PCA) is a technique first described by Pearson (1901) used to emphasize patterns in data while removing emphasis from randomness in the data. It is also used to dimensionally reduce a dataset by allowing it to be transformed into a space where dimensions are aligned with the variance in the data. Dimensions that contribute the least variance to the data are minimized and can be ignored. PCA is similar to a linear regression

except that a linear regression is performed with respect to an independent variable where as PCA is multivariate and calculates a “best fit” describing all dimensions.

By applying Principal Component Analysis to assist in log analysis, the method proposed in this paper has the potential to greatly improve the effectiveness and timeliness of analysis by identifying known events and states in devices.

1.2 Problem Statement

With the increasing use of technology in business processes and the advent of the Internet of Things comes the need to monitor these technology systems. The use of log messages in monitoring is an established, effective, and sometimes, the only method of gaining insight into the health and function of a system. The manual monitoring of log messages becomes more impractical as the volume of logs increases, obscuring important data and overwhelming the analyst. An autonomous method of log analysis is needed that can determine the health state of a monitored system or device to provide a notification of problems.

1.3 Hypotheses

Hypothesis 1 (H1): Principal Component Analysis can be used to characterize system health states.

Hypothesis 2 (H2): The representative principal components can be used to characterize the system health state of a wireless access point from log messages.

Hypothesis 3 (H3): The method of characterization by Principal Component Analysis can be used to identify known failure states in wireless access points when the failure state has been seen before and log data generated during the event exists in the training data.

Hypothesis 4 (H4): The method of characterization by Principal Component Analysis can be used to identify a wireless access point that is overloaded by a high number of connected clients.

Hypothesis 5 (H5): The Principal Component Analysis characterization of a wireless access point can be used to identify unknown failure states from the log messages.

1.4 Definitions

Covariance – Covariance provides a measure of the strength of the correlation between two or more sets of random variates. (Weisstein, 2016)

Eigenvector –A non-zero vector, ν, corresponding with an eigenvalue, λ, which satisfies the linear equation $(A - \lambda \mathrm{I})\nu = 0$ where A is a square matrix. Also known as a characteristic vector.

Eigenvalue – The corresponding scalar, λ, associated with an eigenvector.

Log Aggregator – A host or system that receives syslog messages from numerous systems.

Principal Component Analysis – A multivariate statistical analysis technique used to dimensionally reduce data in order to examine its underlying structure and patterns.

Syslog – A structured message format standardized in RFC 5424 used to convey and record system log information.

Wireless Access Point – A network device bridging Ethernet and 802.11 wireless networks, allowing wireless clients access to the Ethernet network.

1.5 Justification

The volume of log data produced in an enterprise environment and the importance of the systems generating those logs continues to grow. For this data to be useful it must be analyzed, often by hand, a task made more difficult by the large volume of data. A system to provide real time, actionable analysis based on machine learning would reduce the burden of monitoring and analysis of these systems.

1.6 Summary of Proposed Methodology

In order to test the hypotheses, a dataset of log messages will be collected from a research partner with a large number of log producing devices. A learning and classification algorithm will then be implemented using principal component analysis. This algorithm will then be validated using manually classified data from within the sample dataset. The algorithm will be applied to unknown data in order to test classification of hereto-unknown data.

1.6.1 Collection of Log Data

A research partnership with Brigham Young University's Office of Information Technology (BYU OIT) provides access to a log aggregation and search service with log messages from a large number of wireless access points. By using the OIT record of wireless access point issues, instances of access points in states of normal and abnormal operation were identified. Additionally, manual analysis and classification was used to identify and classify instances of particular system states and events of interest. The logs leading up to these instances are used as training data to generate basis features and allow the classifier to identify similar instances.

1.6.2 Implementation of Learning and Detection Algorithm

The classifier uses principal component analysis to calculate a set of basis vectors, the principal components, that best describe the variance in the training data. The resulting components are used to classify the logged event relative to the training data. The detection method used is novel in its application, allowing for existing knowledge to serve as a basis for the work and proofs presented by others to demonstrate the mathematical truth of the approach. The validity of the detection algorithm will be verified with a generic, numerical dataset prior to it being applied to log data.

1.6.3 Creation of Summary Vectors

Using the existing log management system to obtain information about the log messages received, summary vectors will be created that summarize groups of messages within a given time frame. These summary vectors will consist of information, such as message severity, extracted from the messages. The fields the summary vectors will be chosen to avoid specific information that might bias a test. In a further attempt to avoid biased result the field counts are divided by the total count of messages within the time frame to maintain a consistent range of values.

A summary vector based entirely on message severity counts might take the following form for a group of 30 messages with 23 messages with a severity of six, five with a severity of five, and two with a severity of three.

$$v = [0 \quad 0 \quad 0 \quad \tfrac{1}{15} \quad 0 \quad \tfrac{1}{6} \quad \tfrac{23}{30} \quad 0] \tag{1-1}$$

1.6.4 Verification of Learning and Detection Algorithm

Using a training set created through manual classification, the classifier is trained on known data. To confirm correct operation, the classifier then classifies the training data. A feedback loop, manual or otherwise, is used to correct and tune the classifier.

The trained classifier is then presented with data that has been manually classified but is absent from the training data. This allows for further verification as well as determination of the detection rates.

1.6.5 Application of Learning and Detection Algorithm to Live Data

The classifier will be applied to unclassified data. This allows the classifier to be demonstrated in a real world situation.

1.7 Scope

This research will be limited to the analysis and classification of syslog data generated by Xirrus wireless access points.

1.8 Delimitations

The following are outside the scope of this research.

1.8.1 Log Generation

The log data used in this research is defined by the manufacturer of the equipment generating the log messages. The generation and format of these messages is beyond the scope of this research.

1.8.2 Data Indexing, Storage, and Search Engine

The topic of indexing, storage, and search engines for log data is outside of the scope of this research. The described classification engine is abstracted from the log aggregation system.

2 LITERATURE REVIEW

2.1 Log Analysis Through Machine Learning

Veeramachaneni et al. (2016) demonstrated the effective use of supervised machine learning in the detection of cyber attacks. Their AI2 system used an analyst-in-the-loop to classify a subset of events, providing feedback for the machine-learning algorithm. Using this system, AI2 was able to achieve > 86% detection rate and a 4.4% false positive rate.

The approach used by AI2 utilizes an analyst in the loop feed back mechanism. The analyst is presented with a sample set of outlier events, which are classified, or labeled, by the analyst. These classified events are feed back into the learning model. This model is then used to detect events and create a subset of events to be presented to the analyst as part of the next iteration.

2.2 Principal Component Analysis

There exist methods of statistical analysis that allow for insight without the necessity of expertise in a system. One such method is that of Principal Component Analysis. Principal Component Analysis, or PCA, has long been used to simplify and gain insight into multidimensional dataset.

Principal Component Analysis is a technique used to emphasize patterns in data while removing emphasis from randomness in the data (Pearson, 1901). It is also used to dimensionally

reduce a dataset by allowing it to be transformed into a space where dimensions are aligned with the greatest variance and dimensions that contribute the least variance to the data are minimized and can be ignored. PCA is similar to a linear regression except that a linear regression is performed with respect to an independent variable where as PCA is multivariate and calculates a "best fit" describing all dimensions.

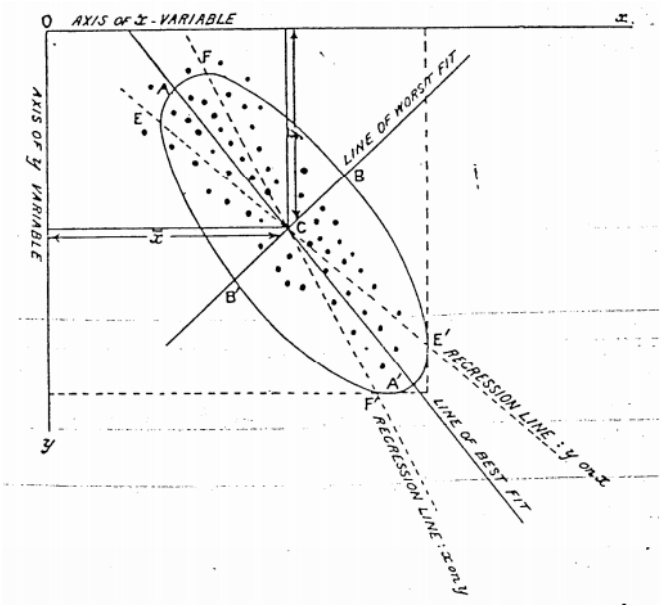

Figure 1 Plot Showing Comparison Between Regression Lines and Lines of Best Fit

Through PCA the data can be expressed as linear combinations of principal components. These principal components represent and are aligned with the variance in the data, with the most significant of the principal components representing the greatest amount of variance. The subsequent principal components each represent the highest possible amount of variance while orthogonal to the preceding principal components. The lesser principal components often represent considerably less variance than the more significant principal components. Because these lesser principal components represent the smallest amount of variance in the original data

they can often be ignored and the remaining components used to represent the original data with a high degree of fidelity in a lower dimensional space.

To demonstrate the steps of PCA the 2 dimensional example dataset X is used. The example dataset was created so that the points would all lie along a line, with slight variations for illustrative purposes.

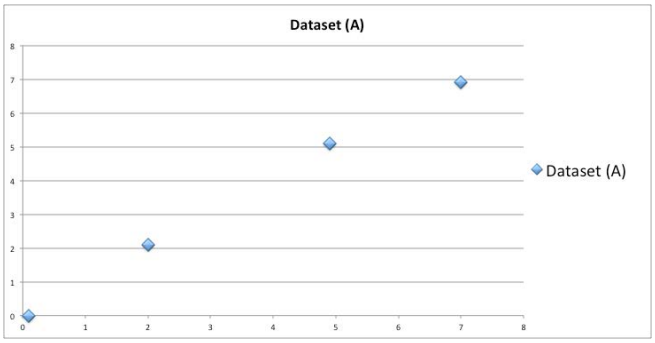

Figure 2 Plot of Example Dataset X

$$\vec{x}_1 = \begin{bmatrix} .1 \\ 0 \end{bmatrix} \vec{x}_2 = \begin{bmatrix} 2 \\ 2.1 \end{bmatrix} \vec{x}_3 = \begin{bmatrix} 4.9 \\ 5.1 \end{bmatrix} \vec{x}_4 = \begin{bmatrix} 7 \\ 6.9 \end{bmatrix} \tag{2-1}$$

$$X = \begin{bmatrix} .1 & 0 \\ 2 & 2.1 \\ 4.9 & 5.1 \\ 7 & 6.9 \end{bmatrix}$$

The mean of each variable in X is calculated and subtracted from each element of that variable, leaving the dataset whose mean value is at the origin. In the example this centered dataset is A.

$$A = \begin{bmatrix} -3.4 & -3.525 \\ -1.5 & -1.425 \\ 1.4 & 1.575 \\ 3.5 & 3.375 \end{bmatrix} \tag{2-2}$$

From a set of vectors arranged in an $m \times n$ matrix, 4x2 in the example dataset, a covariance matrix, S, is calculated. The covariance of two variables is a measure of how much those variables change together and can be calculated with the following:

$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$
$$where\ \bar{X}\ and\ \bar{Y}\ are\ the\ means\ of\ X\ and\ Y\ respectively \quad (2\text{-}3)$$

The sample dataset has the following covariance matrix, S:

$$S = \begin{bmatrix} 9.34 & 9.38 \\ 9.38 & 9.4425 \end{bmatrix} \quad (2\text{-}4)$$

Using the covariance matrix, S, the eigenvectors, v, and eigenvalues, λ, are found by solving the linear equation:

$$(S - \lambda \mathrm{I})v = 0 \quad (2\text{-}5)$$

The resulting eigenvectors point in the directions of variance in the original dataset, and are the components of the data. The resulting eigenvalues indicate the amount of variance accounted for by the corresponding eigenvector. Solving for the eigenvectors and eigenvalues for the covariance matrix, S, from the example dataset results in the following:

$$v_1 = \begin{bmatrix} 0.7051 \\ -0.7090 \end{bmatrix}, v_2 = \begin{bmatrix} 0.7090 \\ 0.7051 \end{bmatrix}, \lambda = \begin{bmatrix} 18.7714 \\ 0.0111 \end{bmatrix} \quad (2\text{-}6)$$

The eigenvectors form the basis for the space that data will be projected onto, with the axis of the eigenspace aligned with the variability in the original data. Using a projection matrix, W, derived from the eigenvectors, the centered example data, A, can be projected onto the eigenspace, resulting in B.

$$B = A \times W$$

$$W = \begin{bmatrix} 0.7051 & -0.7090 \\ 0.7090 & 0.7051 \end{bmatrix}, B = \begin{bmatrix} -4.8969 & -0.0750 \\ -2.0681 & 0.0586 \\ 2.1039 & 0.1179 \\ 4.8610 & -0.1016 \end{bmatrix} \quad (2\text{-}7)$$

When the example data is projected without reducing the dimensionality of the eigenspace the original data can be recovered through an inverse transform.

However, a benefit of PCA is the ability to reduce the dimensionality of data while retaining the ability to accurately represent the data. To demonstrate this the example data, for which the first principal component represented far greater variability than the second, is transformed onto a 1-dimensional eigenspace, resulting in B.

$$W = \begin{bmatrix} 0.7051 \\ 0.7090 \end{bmatrix}, B = \begin{bmatrix} -4.8969 \\ -2.0681 \\ 2.1039 \\ 4.8610 \end{bmatrix} \quad (2\text{-}8)$$

The data is now represented in relation to an eigenvector that in the original space pointed in a direction roughly in line with the original 2-dimensional example dataset. Attempting to recover the original values through an inverse transformation results in the matrix C. The values of C are plotted alongside the original dataset, A, in figure 3.

$$A = \begin{bmatrix} .1 & 0 \\ 2 & 2.1 \\ 4.9 & 5.1 \\ 7 & 6.9 \end{bmatrix}, C = \begin{bmatrix} 0.0468 & 0.5289 \\ 2.0416 & 2.0586 \\ 4.9836 & 5.0167 \\ 6.9279 & 6.9716 \end{bmatrix} \quad (2\text{-}9)$$

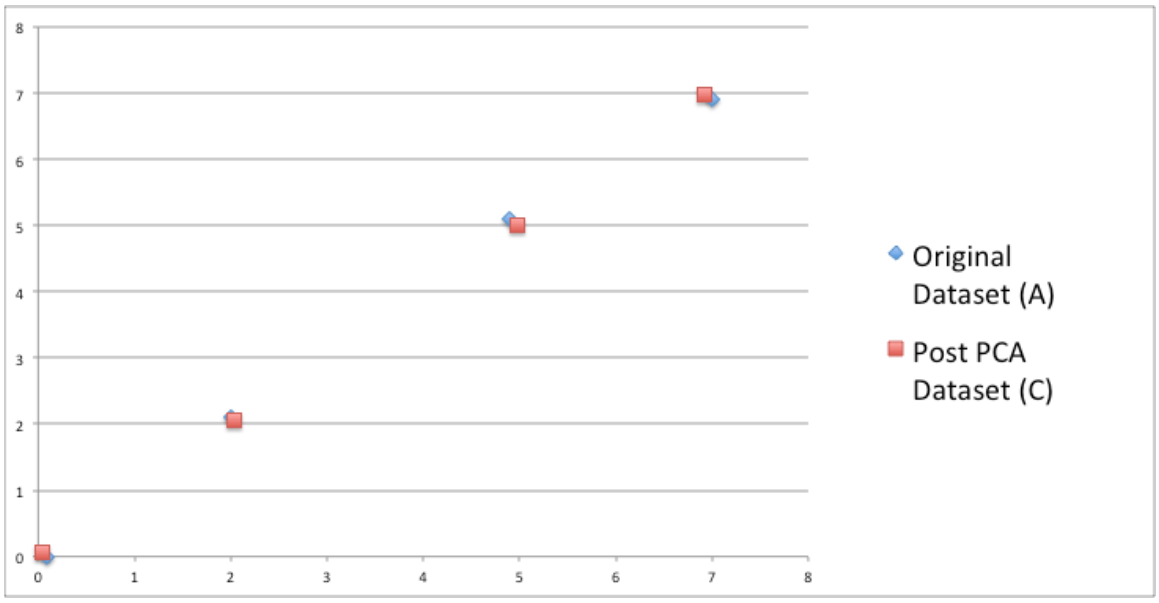

Figure 3 Plot of Original Example Data Before and After PCA

The resulting dataset, C, was represented by a reduced number of dimensions while introducing only a small amount of error. PCA can be applied to larger dimensional datasets to allow for dimensional reduction and gain insight into the data. An example of PCA being used on a 3-dimensional dataset to gain insight into the data is figures 4 and 5, where a plane is generated from the principal components in the directions of greatest variance in the data.

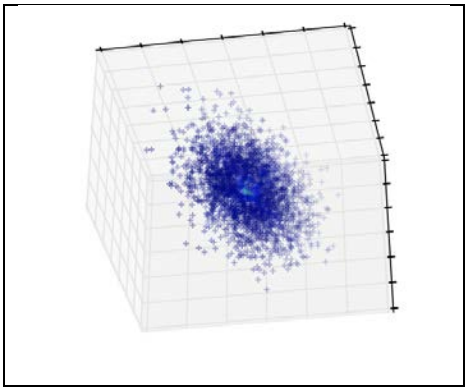

Figure 4 Plot of Large, Seemingly Random, 3 Dimensional Dataset

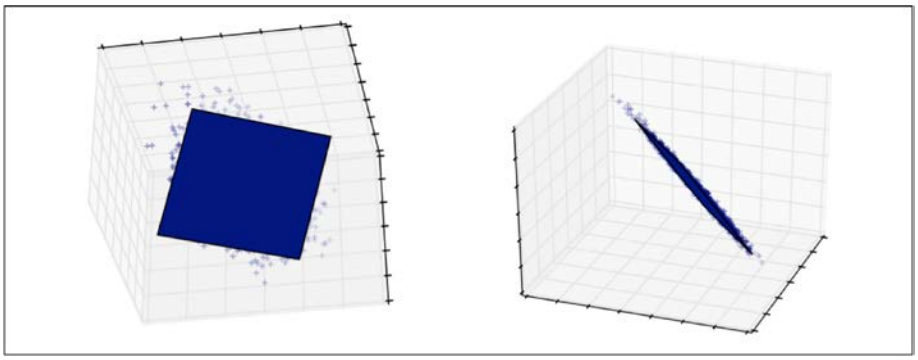

Figure 5 Plot with Plane Defined by the First 2 Principal Components of the Dataset

Alternatively, it is possible to use Singular Value Decomposition to calculate the eigenvectors and eigenvalues directly from the centered dataset. Singular Value Decomposition yields an identical result and is used for its efficiency by many PCA libraries. The library utilized by the proof of concept system uses SVD.

As a result of PCA the original data is now represented by underlying relationships in the data. Furthermore, by eliminating minor components we are able to represent the data in a lower dimensional space while choosing the amount of variance in the original data we want to represent.

2.3 Eigenfaces

The use of principal component analysis, or PCA, for anomaly detection and classification is not a novel approach. Existing applications of the analysis method have proven effective both within information technology and in other fields. The existing applications and the generality of PCA provides confidence that PCA can successfully be applied to characterizing system log data.

A notable example of the effectiveness of PCA is that of eigenfaces for facial characterization, storage, and identification. In the seminal paper “Low-dimensional procedure for the characterization of human faces” (Sirovich & Kirby,1987) it was demonstrated how PCA can be used to characterize and compress an image of a human face. By using PCA on a collection of face images, a set of basis features were formed and used to represent the face image as a low dimensional vector. The computed bases and vector could then be used to reconstruct the original image with a high degree of accuracy.

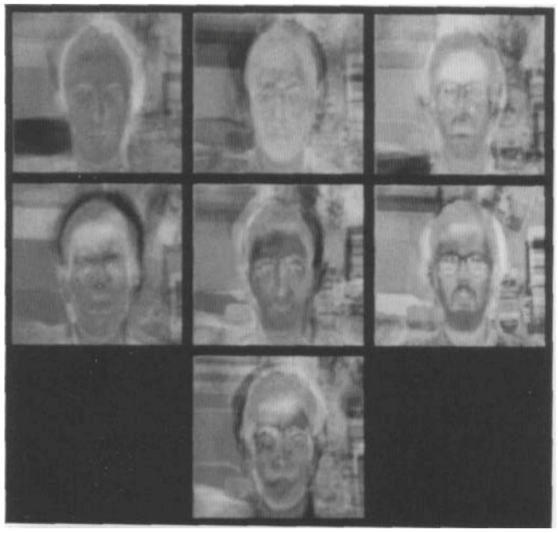

Figure 6 A Sample of Eigenfaces with Each Face Representing a Component in Cigenspace

The PCA approach to representing faces was later applied in “Eigenfaces for recognition” (Turk & Pentland, 1991a) to facial recognition. By using a standard set of bases, referred to as reference faces, it was shown that a previously seen individuals face could be identified and reconstructed even while partially obstructed. The ability to identify an individual when shown an image that is approximately the same, obstructed, and containing other noise, shows the strength of the approach as a method of characterization.

The eigenfaces approach to facial recognition as described by Turk and Pentland (1991b) has two main stages, initialization and recognition. The initialization, or training stage requires that a training set of images be acquired. The eigenfaces, each representing a component in eigenspace, of this training set are calculated and the eigenfaces pruned to the number of dimensions desired. Images of known individuals are then projected into the space defined by the remaining eigenfaces, recording the results as weights of the eigenfaces.

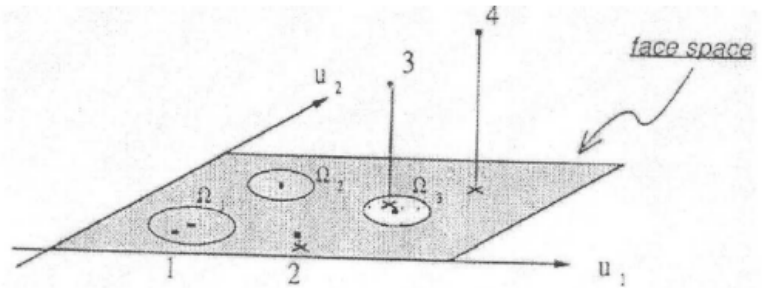

Figure 7 A Simplified Version of Face Space to Illustrate the Results of Projecting

The recognition step is performed by first projecting an image onto the space (Fig. 7) defined by the M eigenfaces. Face recognition is then achieved by calculating the Euclidean distance to the nearest neighbors in eigenspace.

This approach also demonstrates some computational advantages. By using PCA to dimensionally reduce the data, an image in the case of eigenfaces, the computational requirements of classification are greatly reduced. The computational requirements are such that Turk and Pentland note that the training steps could "be performed from time to time whenever there is free excess computational capacity"(1991a). Similarly, by representing the data as a low dimensional vector, the resources required for storage and transport are reduced.

2.4 Stock Market Analysis

Another notable example of the use of PCA and eigenvectors is in characterizing the behavior of the stock market, as demonstrated in "Quantifying and interpreting collective behavior in financial markets." (Gopikrishnan, Rosenow, Plerou, & Stanley, 2001) It is shown that by analyzing and representing the trading behavior of individual stocks as eigenvectors that stocks could be partitioned into market segments with a high degree of accuracy. Stocks of distinct market segments (e.g., Paper or banking) could be identified by eigenvectors that would deviate from the normal for stocks of that market segment. It was also shown that in the analysis performed that there were eigenvectors that were correlated with market capitalization and portfolio returns.

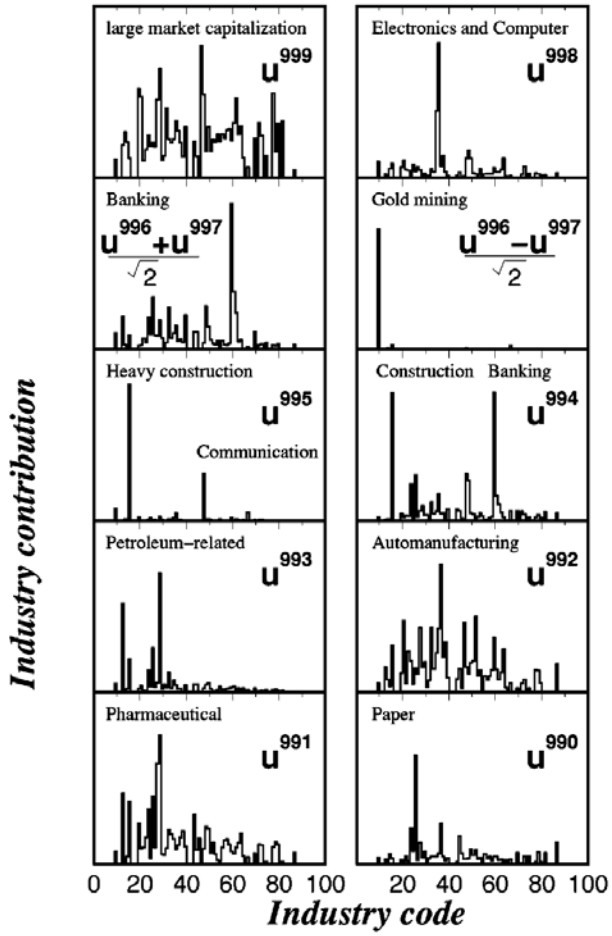

Figure 8 Contribution to Industry Sector of Eigenvector u For The Deviating Eigenvectors

2.5 Analysis of Machine Generated Data

PCA has been applied to problems within the domain of information technology. When PCA is applied to log data it is often to a specific, structured, type of data, such as NetFlow. The use of structured data lends itself more easily to the techniques such as PCA. PCA can also be applied to non-specific data in "Mining Console Logs for Large-Scale System Problem Detection".

In "Mining Console Logs for Large-Scale System Problem Detection" (2008) Xu et al. combine text mining, PCA, and decision trees to detect anomalies in logs and then present them

to a human operator. The goal of their system is to provide a systematic method of detecting important information amongst the large amount of data contained in log messages, especially in large systems.

Their system uses PCA as an intermediate step to perform anomaly detection. In order to perform the anomaly detection, the algorithm is provided with vectors representing message counts corresponding with a specific message format. The result of PCA on the message vectors is then used for anomaly detection based on their dimensionality.

A space is defined that spans all message formats for the software being observed. In order to enumerate all possible messages, the authors examined the source code of the program being monitored. By examining the source, it is possible to determine every possible message that might be generated, along with variable values and meanings. This comprehensive approach ensures that every possible message is accounted for but requires extensive interaction.

While the authors present the system as one that does not require prior input from the operator, this system would still require someone with knowledge of the system to group the messages before the system could be used.

3 METHODOLOGY

3.1 Extend Current Technologies

The current systems for log aggregation and analysis are highly effective at presenting data to analysts and performing basic analysis to be consumed by an analyst. These systems are increasingly complex and require skilled operators to interpret the results.

By extending the methodology used for face detection and recognition in "Eigenfaces for recognition" (Turk & Pentland, 1991a), the proposed system will allow for the detection and classification of system events. In doing so, the burden of the analyst will be reduced and automated detection and response is facilitated.

The proposed system improves on the work of Xu et al. (2008) in allowing for greater flexibility in the system being monitored. Xu et al. utilized source code analysis in order to fully enumerate and account for all possible messages. By instead utilizing generic features from log messages, the proposed system is able to be implemented without a time consuming analysis of source code while also being able to adapt to changes in log messages and formats.

Instead of attempting to implement the classification engine within a particular log aggregation system or attempting to implement a log aggregation system, the classification engine consumes data from a separate log aggregation system. By abstracting the classification

engine from the log aggregation system the classification system becomes log aggregator agnostic.

3.2 Classification Engine

The classification engine analyzes log data and attempts to classify the device state based on training data. It consists of two main components, the trainer and the classifier. Both components use, as input, a vector referred to as the summary vector, which summarizes syslog message feature counts over an arbitrary time window.

3.2.1 Summary Vector Construction

Prior to analysis, data must be represented as a vector. To construct this vector, log messages are queried by source device in five-minute buckets. The messages contained in these buckets undergo feature extraction to extract specific attributes of the data that are then summarized as counts for the time bucket.

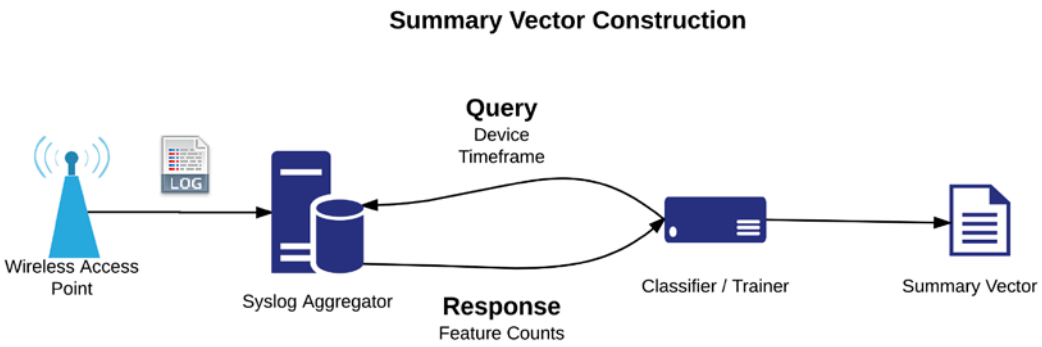

Figure 9 Diagram of the Summary Vector Construction Process

The extraction of features is performed by the log aggregation system at ingestion and stored with the message. This allows for the counts to be aggregated by the log management system and a summation of feature counts returned for a time bucket as part of a query.

The sum of these features is used to construct a vector representing the time bucket and device pair. These summary vectors are used in both the training and characterization steps. In the proof of concept system, the time-framed summaries are represented as 30 dimension vectors consisting of summary counts of the features for the device in a five-minute time window.

The summary vector is constructed from counts of the properties of the messages as follows.

Table 1 Summary Vector Fields

Property	**Dimensions Contributed**
Counts of the 8 severity levels as defined in RFC 5424 (Gerhards, 2009)	8
Counts of messages associated with each of up to 8 internal access points in the wireless access point	8
Count of all messages associated with an internal access point	1
Count of all distinct wireless clients, identified by MAC address, recorded in messages	1
Count of client authentications	1
Count of client deauthentications	1
Count of client associations	1
Count of client association related messages that occurred with the exception of association messages	1
Count of client disassociations	1
Count of client disassociation related messages that occurred with the exception of disassociation messages	1
Count of client reassociations	1
Count of client reassociation related messages that occurred with the exception of reassociation messages	1
Count of client timeouts	1
Count of signal level warning messages	1
Count of rate change messages	1
Count of IDS events reported by the access point	1

A combined count of all severity levels, effectively a count of all log messages is used to normalize the vector. The purpose of the normalization is to minimize the perceived differences between two devices behaving similarly but generating different amounts of messages.

Due to the initial lack of classified vectors, it is necessary for a training step where some are classified manually. By identifying some summary vectors as representing the device during a known problem or state, this information can then be used in the training and classification steps. It is also possible, once a single vector has been classified, to utilize the classifier's ability to detect outliers to quickly train the classifier using analyst feedback.

3.2.2 Trainer

The trainer must perform two actions. First it must perform PCA on a large number of summary vectors in order to calculate the principal components of the data. This step only needs be performed once. Second is the transformation of the training data, consisting of manually classified summary vectors, into the eigenspace computed as part of PCA.

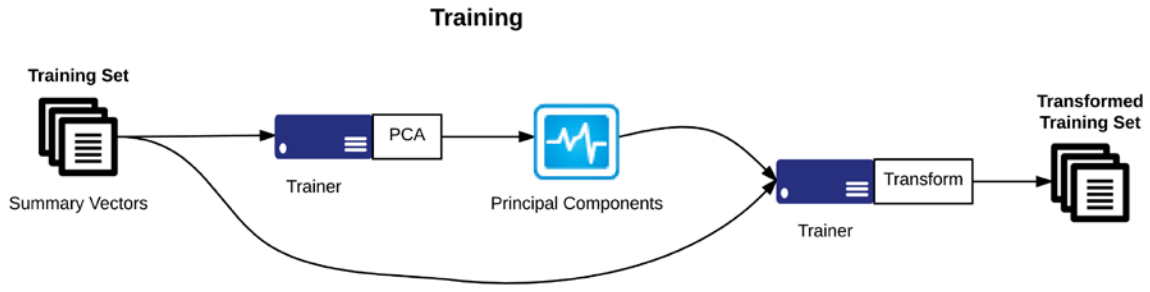

Figure 10 Diagram of the Classifier Training Process

The performing of PCA is by far the most computationally intensive step. To train the classification engine, an $m \times n$ event summary matrix is created consisting of n-dimensional

summary vectors. The row vectors of the matrix contain all summary vectors for the test being run. The use of a larger number of vectors allows for a greater amount of variability to be represented by components resulting from PCA. The principal components are computed in order to determine a set of vectors that represent the variability in the original data. These components provide us with a set of basis vectors that we can use to transform the summary vectors onto a space where they can be more easily represented. By eliminating the components that represent the least amount of variance, the dimensional complexity of the transformed data can be reduced. To train the classifier, the manually classified vectors are transformed to project the vectors onto the eigenspace defined by the components. These known transformed vectors are preserved to be used later by the classifier.

3.2.3 Classifier

The classification engine analyzes log data and attempts to classify the device state based on training data. This is accomplished by transforming the summary vectors from a device onto the eigenspace computed by the trainer. The summary vector is then classified using the Euclidean distance to the nearest training data vector in the eigenspace. The summary vector is then classified as the same class as that of the nearest training points.

The Euclidean distance is also recorded, allowing for filtering and outlier detection. By setting a threshold for classification, the sensitivity of the classifier can be adjusted.

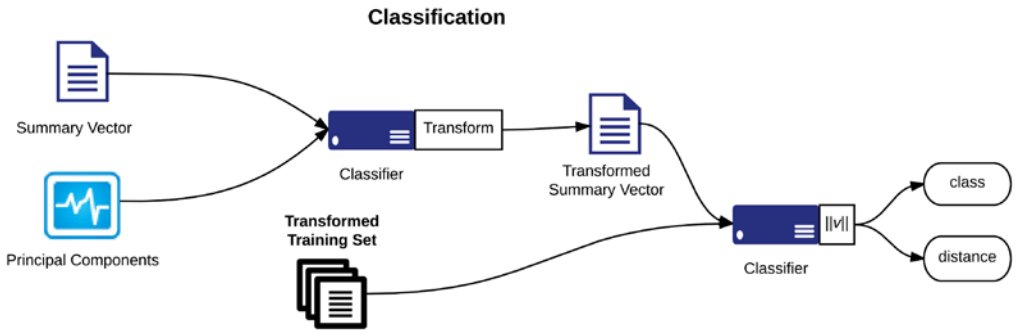

Figure 11 Diagram of the Classification Process

3.2.4 Log Parsing

Log message formats are varied even when conforming to RFC 5424 (Gerhards, 2009). Software changes, or rare events, can generate messages that a log analysis system or analyst might not be able to interpret. The method proposed by Xu et al. (2008) requires that the full message space be enumerated through source code analysis and would be unable to utilize unknown messages. The source code of any monitored process would need to be reviewed after patching or upgrade to ensure messages continue to match the formats known to the system.

The proposed method allows for meaning to be extracted from both common attributes of log messages as well as obscure or even unknown messages, without having to have expert knowledge of the system. In order to do this, log messages must be parsed to obtain enough message attributes to contribute to the summary vector in a meaningful way.

A typical log message from a XIRRUS access point may be similar to the following:

<15>Apr 11 12:41:08 FAUX-345-678-AN1 : debug : Station ee:e6:ad:ff:e1:22 (Intel), IAP iap2: authentication packet sent (3-1)

Using both well known log format standards and context, this log message can be broken down into a summary that can be combined with information from both similar and dissimilar log messages to create a meaningful summary vector.

Roughly following the format defined in RFC 5424, the message can be separated into a header section and a message section. The header contains the message priority, message timestamp, and hostname of the log source. The message section contains information from the application in a format defined by the application creators. In this particular log message from a wireless access point, the message section contains the message severity, records the MAC address of a wireless client and the internal access point on the device it connected to, and finally the action performed by or on the client that caused the message to be generated.

The developers of the application define the message format, which presents the possibility that messages with unknown fields may be observed. To be able to account for these rare occurrences without knowing their format, fields from both the message section as well as the syslog header are used to generate summaries. The fields used were selected by observing a large amount of data and choosing fields that were universally present in a message, such as severity, often present in a message, client MAC address, and rarely present, an IDS alert. A single message could potentially be accounted for in multiple elements of the summary vector.

Ultimately, the number of features used to perform analysis is variable and up to the discretion of those implementing the classifier. While reducing the number of features used can have a significant impact on performance, the use of PCA provides for a separate means of dimensional reduction allowing for a very large number of features to be represented by a very small number of dimensions, if desired. This allows for flexibility in the implementation of the classification engine to account for storage and computational resource limitations.

By offloading the parsing and extraction to a log aggregation system as part of its normal operation, the desired data can be stored and made retrievable as it is generated, significantly reducing the runtime resources required to perform classification. The use of a log management system also increases the rate at which algorithmic changes can be evaluated.

3.3 Test Environment and Training Data

The test environment is the wireless network infrastructure of Brigham Young University. The training of the classifier requires samples of logs from as wide a range of operating conditions as possible. To obtain a broad range of data, summary vectors for devices covering a five-minute window were calculated for all of 2015 and used to populate the sample space for the PCA step. Training data was then obtained by manually analyzing log data to find periods of time during which the device in question was a particular state of interest for the test. The time periods encompassing the events are manually classified to describe the event recorded and added to the training set for the test.

4 IMPLEMENTATION

4.1 Log Management and Data Storage

The proposed method of message classification requires a method of storing and retrieving log data. However, none is proposed as the proposed method of message classification is intended to be log management system agnostic. In order to satisfy the need for a log management system, the infrastructure currently in place at BYU OIT, comprised of Elasticsearch and Logstash, was used. Elasticsearch and local files were used to store data from intermediate steps and provide fast access to that data. The existing log management system utilizes Logstash to collect and parse incoming logs in real time. The existing parsing rules were left in place and extracted data utilized.

4.2 Computational Resources

The log management system for BYU OIT is hosted in a virtual environment. In order to most efficiently interact with the log management system, all computation was performed in the same virtual environment. The computational resources required to perform the classification were drawn from the available pool of the virtual environment.

4.3 Parsing

The existing log management system was used to collect and parse the incoming log messages. This log management system uses a series of pattern matching steps to classify and

extract useful information from the log messages. This extracted data is stored along with the message in Elasticsearch for later search and recall.

4.4 Summary Vector Construction

The parsing performed by the log management system does not provide all of the features required. In the summary vector construction step, the stored data is queried and processed to obtain the necessary features.

During the initial training phase, summary vectors are generated en masse on all available historical data. Due to the large amount of data stored in the log management system, and in order to reduce the load on the log aggregation system, queries are filtered by building and restricted to a one-hour period. Within this one-hour period, results are separated by device and returned in five-minute buckets.

4.5 Training

During training, the n-dimensional summary vectors previously constructed are queried and formed into $m \times n$ matrix of m summary vectors. The computation of principal components is performed using a Python program utilizing the open source scikit-learn library (Pedregosa et al., 2011). All manually classified summary vectors in the training set are transformed into eigenspace and stored in Elasticsearch or locally for use in the classification process.

4.6 Classification

Once transformed, a new summary vector is compared to training set summary vectors that were previously classified. The Euclidean distance is then calculated to the nearest training set vector in the eigenspace. If the Euclidean distance to the nearest neighbor is below the

threshold for classification, the summary vector is classified as the classification of its nearest neighbor. This is similar to the method used by Turk and Pentland (1991b) to perform face identification.

4.7 Verification

For events that have been manually classified as part of a test set, the classifier performed verification of the results. For events that have not been manually classified, an analyst verified the classification.

4.8 Retraining

Once a data point is verified, it can then be used as part of the training set. Classifying a summary vector allows it to be used in the classification step for future events. During retraining, the calculation of principal components does not need to be repeated and the previously computed principal components may be used.

4.9 Test For Load Detection

The goal of this test was to determine if the classifier can detect a high transient load on an access point. In doing so, the hypothesis that PCA can be used to characterize the health of a system (H1) was tested. The test was also to determine if PCA can be used to identify an overloaded wireless access point (H4).

4.9.1 Sample Set

Data was taken from two wireless access points along popular staircases leading to the BYU-Provo campus. One of these devices, Device A, services part of the main staircase in the

BYU Life Sciences Building, and the other, Device B, the main staircase in the BYU Tanner Building. These access points experience significant increases and decreases in mobile client activity at predictable intervals. Figure 12 shows an instance of Device A experiencing a sudden increase in log activity, coinciding with increased wireless client activity. The several thousand logs generated in the period shown in figure 12 are represented by a small number of summary vectors.

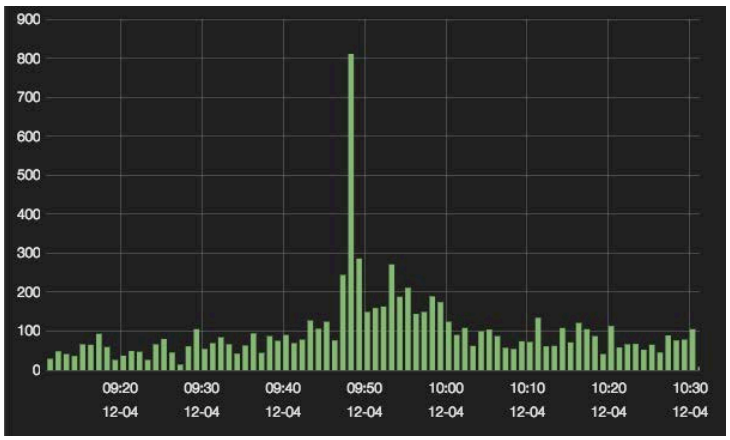

Figure 12 Chart of Log Message Count per Minute on Device Showing Heavy Load

A training set was constructed by manually identifying samples from periods of high load, low load, and from the middle of the night. Of the two access points, Device A was selected to provide the dataset used for training. From this access point, 200 summary vectors were manually classified: 100 were classified as high load, 60 as normal load, and 40 as night load. The classified vectors were split in two, with 100 to serve as training data, and 100 to be used to test the classifier.

From the second access point, Device B, 100 events were manually classified. Of these events, 50 were peak load, 30 normal load, and 20 night load. All of these 100 events were used to test the classifier.

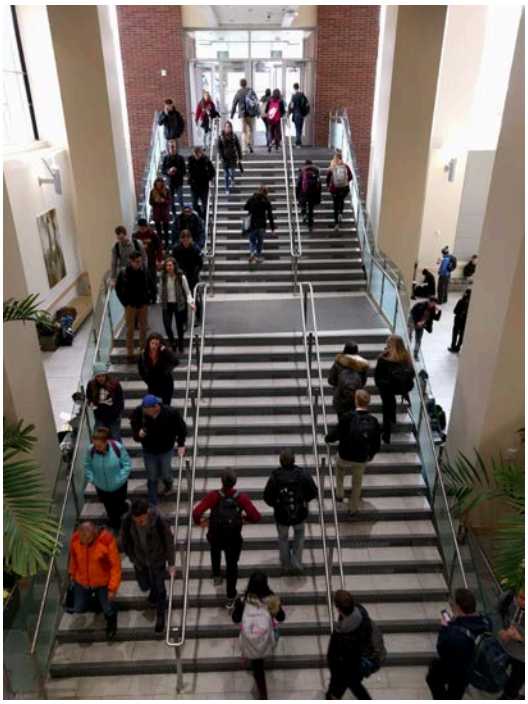

Figure 13 Location of Device A in the Life Sciences Building on the BYU Campus

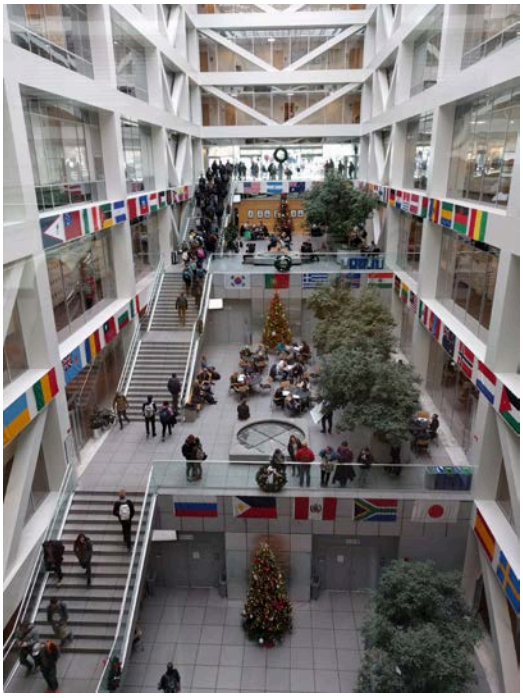

Figure 14 Location of Device B in the Tanner Building on the BYU Campus

4.9.2 Test Procedure

Four separate tests were run in order to examine the characteristics of the classifier and determine how it might function in a multi-device environment. The first tested the classifier under ideal conditions where both the training data and the test data came from the same device. The principal components were calculated using all summary vector data from 2015 for Device A. The classifier was then trained using 100 manually classified summary vectors from Device A. The classifier then attempted to classify the 100 summary vectors in the test set for Device A. The detection threshold of the classifier was set to a sufficiently high value as to allow all summary vectors to be classified.

Due to the initial computational cost of computing the components for a large number of devices, it is desirable that the resulting transform and classifier be portable. Being able to accurately classify summary vectors from previously unseen devices would remove the need to perform PCA on a new dataset and retrain the classifier. To test the portability of a trained classifier to unknown devices, three additional tests were performed that incorporated data from Device B.

The first of these tests attempted to classify a device, Device B, which was unknown to the classifier. The principal components used by the classifier were calculated using a year's work of data from Device A. The training set was also from Device A. The classifier attempted to classify the test data set from Device B.

Next, a combined dataset from both Device A and Device B was used to calculate the principal components used by the classifier. The goal of this test was to determine how additional devices affect accuracy. While a year's worth of summary vectors from both devices

was used when computing principal components, only training data from Device A was used. The classifier attempted to classify the test data from Device A.

Finally, a test was run to determine the accuracy of the classifier from the previous test when classifying test data from Device B.

Table 2 Test Parameter Matrix

Test #	**Device used to compute principal components**	**Device trained on**	**Device classified**
1	A	A	A
2	A	A	B
3	A,B	A	A
4	A,B	A	B

4.9.3 Verification

The classifier results were evaluated with respect to both the accuracy of the classifier and the effect of varying the number of components used on the accuracy of the classifier. Utilizing a known test set allows the true and false classifications to be determined by the classifier, providing a measure of accuracy. The accuracy is defined as the sum of true positive and true negative results over the total number of results. By applying the classifier for varying numbers of components, and therefore dimensions, the effect of dimensional reduction on the accuracy of the classifier can be measured.

4.10 Test for Known Failure Detection

The goal of this test is to determine the ability of a trained classifier to correctly classify failure events that have been previously observed. The hypothesis that PCA can be used to characterize the health of a system (H1) was tested. The hypothesis that PCA can be used to identify a known failure state when trained on the failure (H3) was tested.

4.10.1 Sample Set

A failure event was located in the syslog data for which there were multiple occurrences. A single instance of this failure, a failure resulting in the device restarting, was manually classified and used to train the classifier. The classifier was also trained on the 100 summary vector training set used for the previous tests. Six additional instances of the failure event were found and used to verify the results of the classifier.

4.10.2 Test Procedure

The classifier was trained on the 101 summary vector training set representing the 3 original classifications as well as an additional class consisting of a single summary vector from a power failure event. The detection threshold of the classifier was set to a sufficiently high value as to allow all summary vectors to be classified.

The classifier attempted to classify all summary vectors for the access point of interest a total of 73,141 summary vectors. The ability of the classifier to correctly classify known power failures was evaluated to determine the accuracy of the classifier.

4.10.3 Verification

The classifier results were evaluated to determine the classifier's ability to detect similar occurrences of the event on which it was trained. The classifier's detection ability was measured by quantifying its sensitivity, measured as the percentage of actual occurrences of the event correctly detected.

5 RESULTS

5.1 High Load

5.1.1 Sample Set / Training Data

A wireless access point was chosen that regularly experienced a high volume of transient clients due to its location along a major staircase providing access to the BYU campus. This device would experience a peak of activity generally lasting around 10 minutes and beginning around 10 minutes before the hour during the week. Between these peaks and in the evening, the device would experience significantly less activity. From the summary vectors for this access point, 200 data points were manually classified to be used as training and test data: 100 peak load, 60 off peak load, and 40 night load. The data was split in half, with half being the training set and half the test set. With the goal being to identify periods of peak load, half of the events (50 in both the training and test set) were from periods of peak load and half from periods of lesser load.

5.1.2 Computational Requirements

The initial computation of the principal components was the most significant limiting factor to the ability for this technique to scale. This technique relies on a large dataset consisting, in this case, of all available data points in order to ensure that the principal components of the data can accurately be computed. The logs from one of the devices in the dataset resulted in approximately 73,000 summary vectors over the course of a year. To compute the principal

components for a dataset derived from all of the devices on the network, would require a prohibitively large amount of computational resources. The resulting principal components can be reused indefinitely by the classification engine.

The conversion of log data into the five-minute summaries is able to run in real time, if not limited by the log aggregation system.

The CPU and memory requirements for classification are very low and could be easily distributed. The actual classification, performed after the generation of the summary vectors, requires the summary vector be multiplied by the transformation matrix, followed by the Euclidean distance being calculated to transformed training events.

5.1.3 Dimensional Reduction

All summary vectors from 2015, for the device or devices represented in tests, were used to calculate the principal components used to define the eigenspace. The calculation of the principal components was limited to a maximum of 30 components, equal to the dimensionality of the summary vectors. The accuracy of the classifier was evaluated with varying numbers of components.

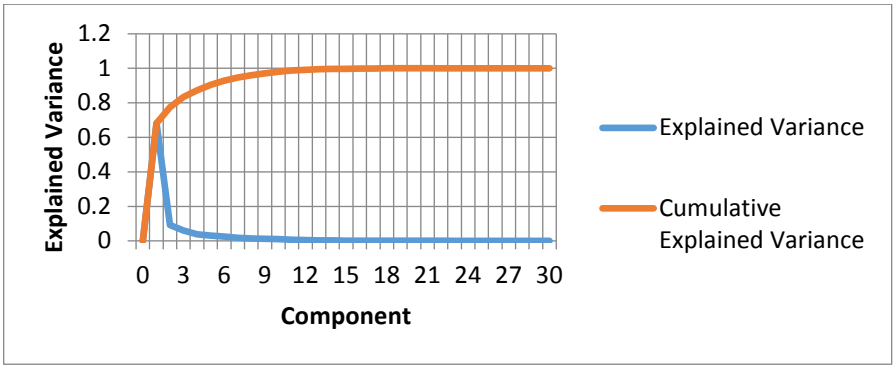

Figure 15 Explained and Cumulative Explained Variance w.r.t. Number of Components

The first principal component in all test configurations represented greater than 60% of the explained variance, with the explained variance represented by subsequent components quickly dropping to insignificant levels, as shown by the explained variance plot in figure 15. By the 25th component, ordered by the variance explained by the component appears to be, at least partially, the result of floating point math error. Additionally, there appears to be no benefit from more than 22 components in any test configuration. In some instances, additional components resulted in increased error, as shown by the decrease in accuracy in figure 16.

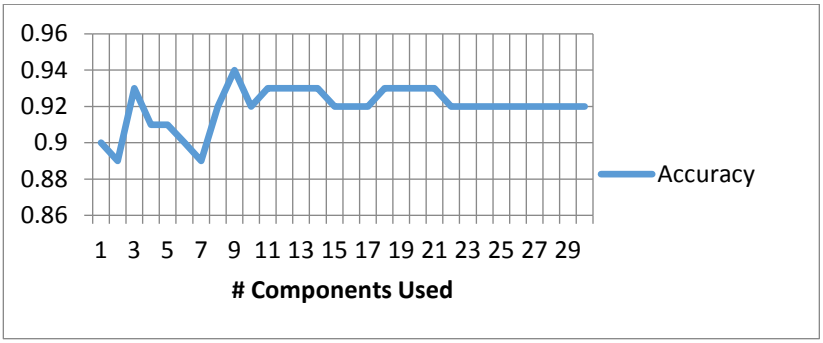

Figure 16 Results of Test 1:Accuracy of Classifier w.r.t. Number of Components

A significantly reduced number of components were found to be able to represent the data without a significant reduction in the accuracy of the classifier. This allowed for a lower number of dimensions than the original dataset to be used to represent the data, reducing storage and computational requirements.

5.1.4 Accuracy in Determining Load

In a test, Test 1, attempting to identify periods of heavy load on a single device, device A, while having trained the classification engine on that same device, the classification engine proved to be accurate. As shown in figure 16, the classification engine was able to consistently

classify a time period as high load or not, with rates of 90% or above when representing data using 8 or more components. When using fewer than 8 components the results fluctuate, reducing confidence in the actual accuracy. In this test, the rates of false positive and false negative classification roughly equal as shown in figure 17.

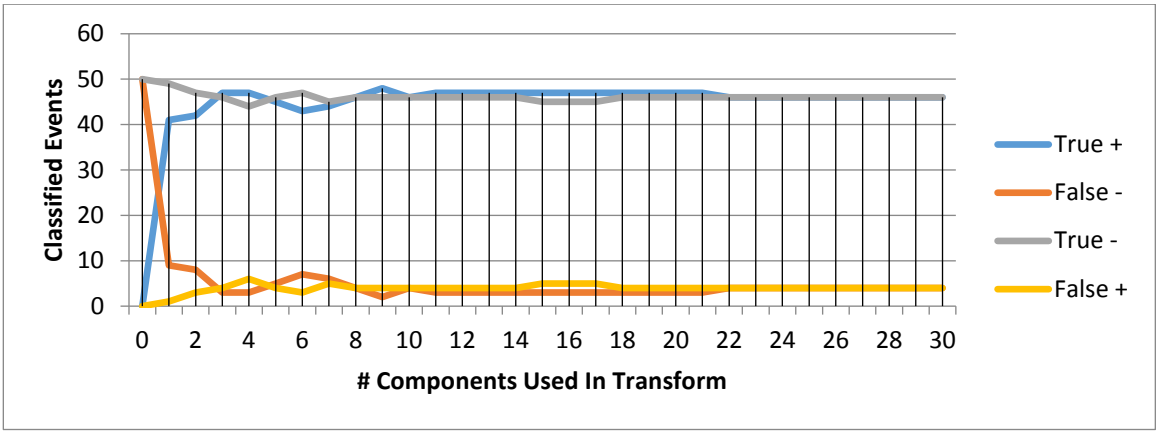

Figure 17 Results from Test 1: Classification w.r.t. Number of Components

5.1.5 General Application of Single Device PCA

It was found that the classifier could be trained on one device and used to classify summary vectors on a second device, previously unknown to the classifier. In Test 2, when used to classify summary vectors from Device B as high load or not, a classifier trained on Device A was able to consistently achieve accurate classification rates of 85% to 86% when representing data using 8 or more components.

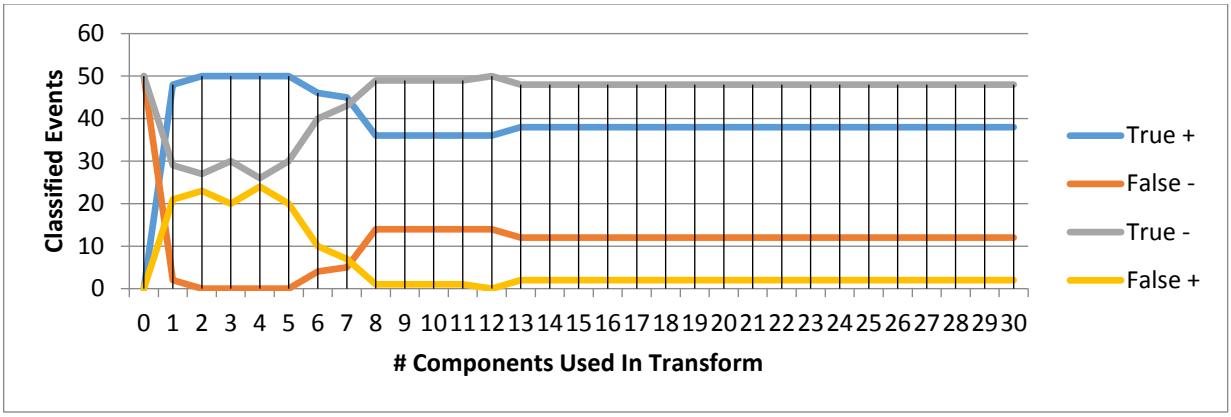

Figure 18 Results of Test 2: Classification w.r.t Number of Components

Test 4, utilizing the summary vectors from both devices when computing the principal components, had similar results to Test 2 when classifying summary vectors from Device B. As shown in figures 18 and 19 the classification rates are similar between Test 2 and Test 4, respectively, when using more than 8 components.

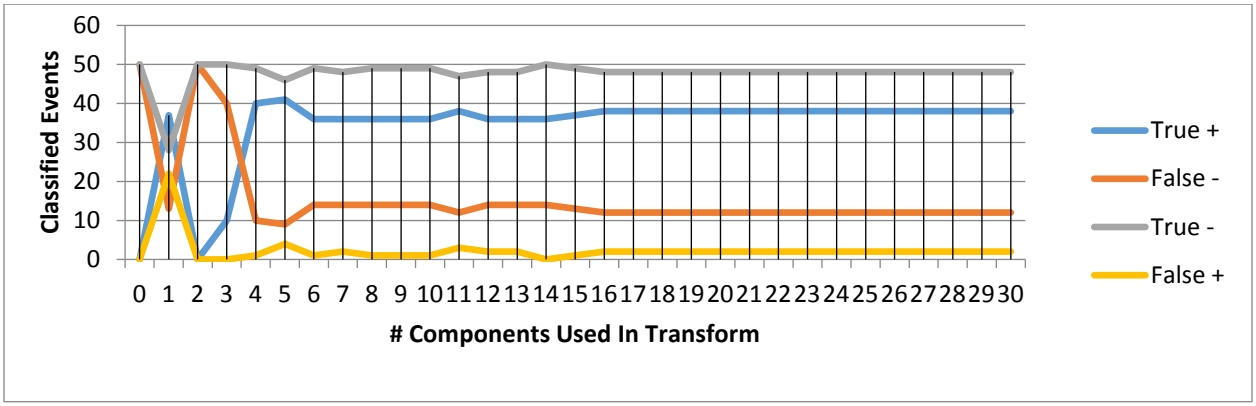

Figure 19 Results of Test 4: Classification w.r.t Components Used

Finally, Test 3, in which data from both devices was used to calculate principal components and summary vectors from Device A were classified, yielded similar accuracy and classification rates to those found in Test 1.

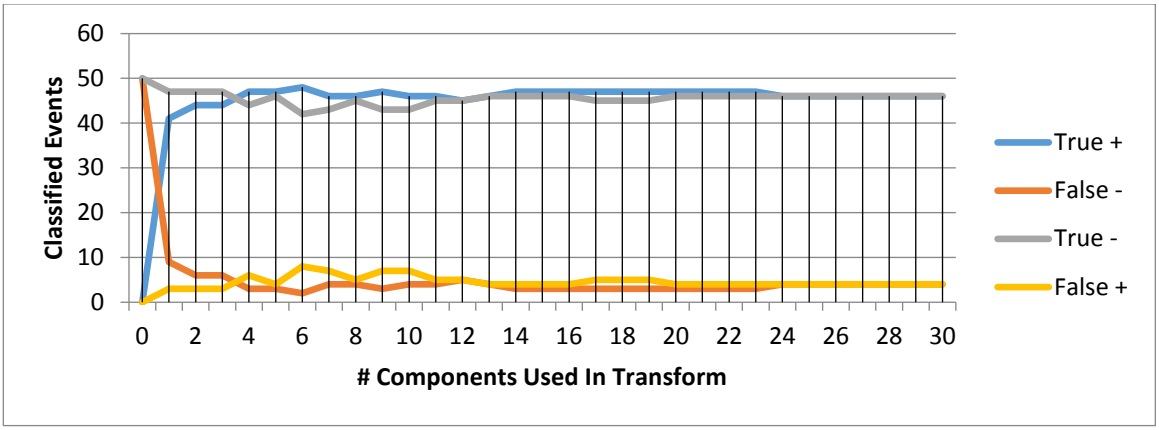

Figure 20 Results of Test 3: Classification w.r.t Components Used

The results of these test show that the classifier is portable in that is can be used to classify summary vectors from unknown devices.

5.2 Identification of Unknown Failures

It was not possible to use the classifier to determine that a device was in an unknown state. While attempting to train the classifier using a manual feedback loop, observations were made about the behavior of the classifier. It was found that the classifier was not effective in detecting when a system is in an unknown state.

When classifying events from the wireless access points, all summary vectors could be classified as either experiencing heavy load or not experiencing heavy load. This is because all devices are in one of the two states at all times. Due to this, the classifier is always able to assign a class to a summary vector that might also represent an otherwise unknown state.

5.3 Classification of Known Failure States

Using a single manually classified instance of a failure state, the classifier was able to identify, with 100% accuracy, similar occurrences of that device state. Out of a total dataset of

73,141 summary vectors, the classifier was able to classify all 6 occurrences of the failure. While the classifier correctly classified all instances of the failure state, the classifier incorrectly classified 603 summary vectors as representing the failure state. This represents a false positive rate of 0.82%.

Table 3 Results of Known Failure Detection

Total Events	**Known Failure Events**	**Correctly Classified**	**Incorrectly Classified**
73,141	6	6	603

6 CONCLUSION

It was shown that a classifier based on PCA could correctly classify a device as being in either a high state of transient load or not in a high state of transient load with a high degree of accuracy, confirming Hypothesis 4. Furthermore, it was shown that a classifier trained on one device could be applied to a second device, unknown to the classifier, and remain accurate.

It was shown that when trained on a known failure event the classifier could identify other instances of the same failure, confirming Hypothesis 3. Without the use of a threshold, the classifier was able to detect other instances of the failure event with a 100% success rate. However, the same test experienced a false positive rate several orders of magnitude greater than the true positive rate. Despite the high false positive rate the ability to detect similar events is confirmed. It is likely that more thorough training of the classifier would decrease the false positive rate.

In confirming Hypotheses 3 and 4, Hypothesis 1, which states that Principal Component Analysis can be used to classify the state of a system, was confirmed.

Hypothesis 2, which states that system states or events could be correlated to individual principal components, was disproven. While Gopikrishnan (2001) was able to correlate eigenvectors from trading activity to market segments, no such correlation was readily apparent for system log data. It is believed that this is in part due to the very narrow scope of the testing performed. The scope of research was such that only a small number of devices and specific

events were examined in detail. To be able to identify such correlations, if they exist, would likely require a much greater survey of the system logs.

Hypothesis 5, which states that unknown failure events could easily be detected, was disproven. The classifier was not able to accurately identify unknown failure states. It is believed that this failure highlights two shortcomings of the methodology. The first shortcoming is the method used to select the training set.

In order to select a training set, the author selected 100 data points from a year's worth of data, that were intended to represent the two classes: heavily loaded and not heavily loaded. This approach resulted in chronologically clustered data points, not necessarily representative of how a classifier might be iteratively trained through analyst feedback in a real world implementation. Unfortunately, machine-learning methods often provide little, if any, insight into how a result was reached, and so the effect of different methods of selecting training data cannot be easily determined.

Another shortcoming of the classifier's ability to detect previously unknown states is the way in which classification is being performed. The binary nature of the classifier allows it to only decide upon one class. When attempting to classify a summary vector as one of two sets into which all vectors fall, such as heavily loaded and not heavily loaded, this shortcoming is not apparent.

This shortcoming became apparent when attempting to detect occurrences of an event. In this set, the training set consisted of 3 classes: loaded, unloaded, and error state. Any given summary vector would, by definition, fall into either the loaded or unloaded class while also potentially being in a detectable error state. When attempting to classify a summary vector, the classifier only allows it to be detected as a single class as the single binary classifier does not

allow for a summary vector to receive multiple classifications. As implemented, the classifier will assign the vector the classification of the nearest summary vector in the training set.

7 FUTURE WORK

Future work could attempt to overcome the shortcomings of the classifier. Multiclass classification is not a new problem. The presented classifier lends itself well to a multiclass classification strategy referred to as one-vs-all.

A classifier using a one-vs-all strategy trains a classifier for each class and attempts to classify a data point against every class individually. The one-vs-all strategy lends itself well to the presented classifier implementation, requiring that multiple, single class classification attempts are made in place of the single classification attempt as implemented.

A further shortcoming of the classifier is the use of Euclidean distance as a classification algorithm. By using Euclidean distance, which classifies based on the nearest neighbor, the classifier is subject to bias from classifications that are more heavily or sparsely represented in the training set. Should an event of class ‘A’, as a result of chance or over representation, be the lowest Euclidean distance from a training set event of class ‘B’, it would be classified incorrectly as class ‘B’. Compounding this problem would be a feedback loop that accepts the incorrectly classified event into the training set, potentially resulting in additional incorrect classifications.

The classifier could also be improved by using an algorithm such as k-Nearest Neighbors, which takes into account multiple nearest neighbors in classification, preventing a single outlier event from causing incorrect classifications.

REFERENCES

Gerhards, R. (2009). RFC 5424: The syslog protocol. *Request for Comments, IETF*.

Gopikrishnan, P., Rosenow, B., Plerou, V., & Stanley, H. E. (2001). Quantifying and interpreting collective behavior in financial markets. *Physical Review E*, *64*(3), 035106.

Hotelling, H. "Analysis of a complex of statistical variables into principal components." *Journal of educational psychology* 24.6 (1933): 417.

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559-572.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825-2830.

Shyu, M. L., Chen, S. C., Sarinnapakorn, K., & Chang, L. (2003). *A novel anomaly detection scheme based on principal component classifier*. MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING.

Sirovich, L., & Kirby, M. (1987). Low-dimensional procedure for the characterization of human faces. *Josa a*, *4*(3), 519-524.

Turk, M., & Pentland, A. (1991a). Eigenfaces for recognition. *Journal of cognitive neuroscience*, *3*(1), 71-86.

Turk, M., & Pentland, A. (1991b). Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on* (pp. 586-591). IEEE.

Veeramachaneni, K., & Arnaldo, I., Cuesta-Infante, A., Korrapati, V., Bassias, C., Li, K. (2016). AI2: Training a big data machine to defend.

Weisstein, Eric W. "Covariance." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/Covariance.html

Xu, W., Huang, L., Fox, A., Patterson, D. A., & Jordan, M. I. (2008). Mining Console Logs for Large-Scale System Problem Detection. *SysML*, *8*, 4-4.

Yambor, W. S. (2000). *Analysis of PCA-based and Fisher discriminant-based image recognition algorithms* (Master's thesis, Colorado State University).

APPENDICES

APPENDIX A. RESULTS TABLES

Test against known device when trained against known device

Components	Events	Explained Variance	Cumulative Variance	True +	False -	True -	False +	Accuracy
0	100	0	0	0	50	50	0	0.5
1	100	0.6817139	0.68171396	41	9	49	1	0.9
2	100	0.0922739	0.77398794	42	8	47	3	0.89
3	100	0.0603007	0.834288729	47	3	46	4	0.93
4	100	0.0377505	0.872039309	47	3	44	6	0.91
5	100	0.0312185	0.903257818	45	5	46	4	0.91
6	100	0.0257975	0.92905541	43	7	47	3	0.9
7	100	0.0168097	0.945865167	44	6	45	5	0.89
8	100	0.0131803	0.959045501	46	4	46	4	0.92
9	100	0.0115934	0.970638967	48	2	46	4	0.94
10	100	0.0103138	0.980952803	46	4	46	4	0.92
11	100	0.0065778	0.987530622	47	3	46	4	0.93
12	100	0.0042791	0.991809763	47	3	46	4	0.93
13	100	0.0025550	0.994364803	47	3	46	4	0.93
14	100	0.0020134	0.996378212	47	3	46	4	0.93
15	100	0.0013944	0.997772698	47	3	45	5	0.92
16	100	0.0008847	0.998657405	47	3	45	5	0.92
17	100	0.0005649	0.99922239	47	3	45	5	0.92
18	100	0.0004228	0.999645256	47	3	46	4	0.93
19	100	0.0001844	0.999829754	47	3	46	4	0.93
20	100	8.84E-05	0.999918139	47	3	46	4	0.93
21	100	6.45E-05	0.999982614	47	3	46	4	0.93
22	100	1.74E-05	1	46	4	46	4	0.92
23	100	9.46E-28	1	46	4	46	4	0.92
24	100	3.08E-28	1	46	4	46	4	0.92
25	100	3.91E-33	1	46	4	46	4	0.92
26	100	3.91E-33	1	46	4	46	4	0.92
27	100	3.91E-33	1	46	4	46	4	0.92

28	100	3.91E-33	1	46	4	46	4	0.92
29	100	3.91E-33	1	46	4	46	4	0.92
30	100	3.91E-33	1	46	4	46	4	0.92

Test against unknown device when trained against known device

Components	Events	Explained Variance	Cumulative Variance	True +	False -	True -	False +	Accuracy
0	100	0	0	0	50	50	0	0.5
1	100	0.6817139	0.68171396	48	2	29	21	0.77
2	100	0.0922739	0.77398794	50	0	27	23	0.77
3	100	0.0603007	0.834288729	50	0	30	20	0.8
4	100	0.0377505	0.872039309	50	0	26	24	0.76
5	100	0.0312185	0.903257818	50	0	30	20	0.8
6	100	0.0257975	0.92905541	46	4	40	10	0.86
7	100	0.0168097	0.945865167	45	5	43	7	0.88
8	100	0.0131803	0.959045501	36	14	49	1	0.85
9	100	0.0115934	0.970638967	36	14	49	1	0.85
10	100	0.0103138	0.980952803	36	14	49	1	0.85
11	100	0.0065778	0.987530622	36	14	49	1	0.85
12	100	0.0042791	0.991809763	36	14	50	0	0.86
13	100	0.0025550	0.994364803	38	12	48	2	0.86
14	100	0.0020134	0.996378212	38	12	48	2	0.86
15	100	0.0013944	0.997772698	38	12	48	2	0.86
16	100	0.0008847	0.998657405	38	12	48	2	0.86
17	100	0.0005649	0.99922239	38	12	48	2	0.86
18	100	0.0004228	0.999645256	38	12	48	2	0.86
19	100	0.0001844	0.999829754	38	12	48	2	0.86
20	100	8.84E-05	0.999918139	38	12	48	2	0.86
21	100	6.45E-05	0.999982614	38	12	48	2	0.86
22	100	1.74E-05	1	38	12	48	2	0.86
23	100	9.46E-28	1	38	12	48	2	0.86
24	100	3.08E-28	1	38	12	48	2	0.86
25	100	3.91E-33	1	38	12	48	2	0.86
26	100	3.91E-33	1	38	12	48	2	0.86
27	100	3.91E-33	1	38	12	48	2	0.86
28	100	3.91E-33	1	38	12	48	2	0.86
29	100	3.91E-33	1	38	12	48	2	0.86
30	100	3.91E-33	1	38	12	48	2	0.86

Test against known device when trained against known and unknown devices

Components	Events	Explained Variance	Cumulative Variance	True +	False -	True -	False +	Accuracy
0	100	0	0	0	50	50	0	0.5
1	100	0.6326627	0.632662717	41	9	47	3	0.88
2	100	0.0932007	0.72586345	44	6	47	3	0.91
3	100	0.0536054	0.77946888	44	6	47	3	0.91
4	100	0.0515263	0.830995207	47	3	44	6	0.91
5	100	0.0430099	0.874005118	47	3	46	4	0.93
6	100	0.0274989	0.901504101	48	2	42	8	0.9
7	100	0.0208315	0.922335651	46	4	43	7	0.89
8	100	0.0176335	0.939969237	46	4	45	5	0.91
9	100	0.0122180	0.952187326	47	3	43	7	0.9
10	100	0.0105326	0.962719999	46	4	43	7	0.89
11	100	0.0094331	0.972153138	46	4	45	5	0.91
12	100	0.0081938	0.980346993	45	5	45	5	0.9
13	100	0.0063329	0.986679983	46	4	46	4	0.92
14	100	0.0035834	0.990263457	47	3	46	4	0.93
15	100	0.0027876	0.993051088	47	3	46	4	0.93
16	100	0.0026667	0.995717868	47	3	46	4	0.93
17	100	0.0012387	0.996956583	47	3	45	5	0.92
18	100	0.0011460	0.998102646	47	3	45	5	0.92
19	100	0.0007410	0.998843694	47	3	45	5	0.92
20	100	6.70E-04	0.999513337	47	3	46	4	0.93
21	100	2.43E-04	0.999756699	47	3	46	4	0.93
22	100	1.41E-04	0.999897441	47	3	46	4	0.93
23	100	8.88E-05	0.999986219	47	3	46	4	0.93
24	100	1.38E-05	1	46	4	46	4	0.92
25	100	2.15E-26	1	46	4	46	4	0.92
26	100	9.03E-28	1	46	4	46	4	0.92
27	100	4.46E-33	1	46	4	46	4	0.92
28	100	4.46E-33	1	46	4	46	4	0.92
29	100	4.46E-33	1	46	4	46	4	0.92
30	100	4.46E-33	1	46	4	46	4	0.92

Detection against unknown device when trained on known and unknown devices

Components	Events	Explained Variance	Cumulative Variance	True +	False -	True -	False +	Accuracy
0	100	0	0	0	50	50	0	0.5

1	100	0.6326627	0.632662717	37	13	28	22	0.65
2	100	0.0932007	0.72586345	0	50	50	0	0.5
3	100	0.0536054	0.77946888	10	40	50	0	0.6
4	100	0.0515263	0.830995207	40	10	49	1	0.89
5	100	0.0430099	0.874005118	41	9	46	4	0.87
6	100	0.0274989	0.901504101	36	14	49	1	0.85
7	100	0.0208315	0.922335651	36	14	48	2	0.84
8	100	0.0176335	0.939969237	36	14	49	1	0.85
9	100	0.0122180	0.952187326	36	14	49	1	0.85
10	100	0.0105326	0.962719999	36	14	49	1	0.85
11	100	0.0094331	0.972153138	38	12	47	3	0.85
12	100	0.0081938	0.980346993	36	14	48	2	0.84
13	100	0.0063329	0.986679983	36	14	48	2	0.84
14	100	0.0035834	0.990263457	36	14	50	0	0.86
15	100	0.0027876	0.993051088	37	13	49	1	0.86
16	100	0.0026667	0.995717868	38	12	48	2	0.86
17	100	0.0012387	0.996956583	38	12	48	2	0.86
18	100	0.0011460	0.998102646	38	12	48	2	0.86
19	100	0.0007410	0.998843694	38	12	48	2	0.86
20	100	6.70E-04	0.999513337	38	12	48	2	0.86
21	100	2.43E-04	0.999756699	38	12	48	2	0.86
22	100	1.41E-04	0.999897441	38	12	48	2	0.86
23	100	8.88E-05	0.999986219	38	12	48	2	0.86
24	100	1.38E-05	1	38	12	48	2	0.86
25	100	2.15E-26	1	38	12	48	2	0.86
26	100	9.03E-28	1	38	12	48	2	0.86
27	100	4.46E-33	1	38	12	48	2	0.86
28	100	4.46E-33	1	38	12	48	2	0.86
29	100	4.46E-33	1	38	12	48	2	0.86
30	100	4.46E-33	1	38	12	48	2	0.86

Detection of known failure events

Total Events:	73,141
Total Classified:	609
Known Similar Events Correctly Classified:	6
Incorrectly Classified:	603

APPENDIX B. SOURCE CODE

Summary Vector Generation

```
#!/usr/bin/python -u
import requests
import json
import datetime
from requests.auth import HTTPDigestAuth

debug=False
count=0

searchHead="172.262.23.0"

workingIndex=None
startTime=None
buildingsList=[]


LSBquery={
  "query": { "match": { "dev.raw": "LSB-1" } }
}



def sendQuery(query, index, count):
        request=json.dumps(query)
        if count == 0:
                url = 'http://'+searchHead+':9200/'+index+'/_search?search_type=count&pretty'
        else:
                url = 'http://'+searchHead+':9200/'+index+'/_search?size='+str(count)+'&pretty'
        r = requests.post(url , auth=('user', 'asdfdfsasdfasdfa'), data=request)
        t=json.loads(r.text)
        return t


temp = sendQuery(LSBquery, "lane-thesis-meta-2015", 200000)
for doc in temp['hits']['hits']:
        print
'['+doc['_id']+','+str(doc['_source']['@timestamp'])+','+str(doc['_source']['bucketTime'])+','+str(doc['_source']['dev'])+','+s
tr(doc['_source']['building'])+','+str(doc['_source']['severityTotal'])+','+str(doc['_source']['S0'])+','+str(doc['_source']['S1'])
+','+str(doc['_source']['S2'])+','+str(doc['_source']['S3'])+','+str(doc['_source']['S4'])+','+str(doc['_source']['S5'])+','+str(do
c['_source']['S6'])+','+str(doc['_source']['S7'])+','+str(doc['_source']['iap1'])+','+str(doc['_source']['iap2'])+','+str(doc['_sou
rce']['iap3'])+','+str(doc['_source']['iap4'])+','+str(doc['_source']['iap5'])+','+str(doc['_source']['iap6'])+','+str(doc['_source'
]['iap7'])+','+str(doc['_source']['iap8'])+','+str(doc['_source']['iapTotal'])+','+str(doc['_source']['auths'])+','+str(doc['_sourc
e']['deauths'])+','+str(doc['_source']['associated'])+','+str(doc['_source']['associationsOverhead'])+','+str(doc['_source']['
disassociated'])+','+str(doc['_source']['disassociationsOverhead'])+','+str(doc['_source']['reassociated'])+','+str(doc['_so
urce']['reassociationsOverhead'])+','+str(doc['_source']['MACs'])+','+str(doc['_source']['IDSevents'])+','+str(doc['_sourc
```

```
e']['rateChange'])+','+str(doc['_source']['signalLevelWarnings'])+','+str(doc['_source']['timeouts'])+']'

exit()
```

Test Run

```
#!/usr/bin/python
import time
import numpy as np
from sklearn import decomposition
from sklearn import datasets

import random

runtype=1
fname="stage1data.csv"
fname2="stage1data.csv"

def arraystr2num(a):
        b=[]
        for val in a:
                b.append(int(val))
        return b

def calcPCA(spaceIn, comps):
      pca = decomposition.PCA(n_components=comps)
      pca.fit(spaceIn)
      return pca


def sortToLists(id, values, time):
        for entry in eMaster:
                if id == entry[0]:
                        elMaster.append([id, entry[1], values])
        if runtype == 2 or runtype == 3 or runtype == 13:
                for entry in testSet:
                        if id == entry[0]:
                                testMaster.append([entry[1],[id, values, time]])

def transformTrainingSet(transMatrix, trainingSet):
        for a in trainingSet:
            tlMaster.append([a[1], transMatrix.transform(a[2])])


def closestGroup(transMatrix, entry):
      bucket= transMatrix.transform(entry[1])
      closest=909999
        closestClass=0
      for a in tlMaster:
            dist = np.linalg.norm(a[1]-bucket)
            if dist < closest:
                  closest = dist
                        closestClass = a[0]
      if closest < threshold:
                if runtype==1:
                        return entry[2]+','+str(closestClass)+','+str(closest)
                if runtype==2 or runtype==3:
                        return [ closestClass, closest ]

fname="stage1data.csv"
```

```
threshold = 99

e1=['AVZ3NxEEWoI242VYAc5w',
'AVZ9ysejNaw7GqRUImyB',
'AVZ-8YcFNaw7GqRUWrDR',
'AVaEOt_uIWASxjuiMPi5',
'AVZ9y_bSNaw7GqRUIqjj']
e2=['AVZ_C0U2I5_rTozncrcU',
'AVZ6E2VAWoI242VYmItD',
'AVZ6E6ulWoI242VYmJjV']

e3=['AVZ3IROaWoI242VYFGQX',
'AVZ96BCNI5_rToznN29T',
'AVZ95_iII5_rToznN2m3']

e4=['AVZ24icNWoI242VY8ZEg']

el1=[]
el2=[]
el3=[]
el4=[]

tl1=[]
tl2=[]
tl3=[]
tl4=[]

eMaster=[]
for a in e1:
        eMaster.append([a,"1"])
for a in e2:
    eMaster.append([a,"2"])
for a in e3:
    eMaster.append([a,"3"])
for a in e4:
    eMaster.append([a,"4"])

elMaster=[]
tlMaster=[]

testSet=[]

test1=['AVZ-8jXTNaw7GqRUWtSK',
'AVaE66RbNaw7GqRUxQL1',
'AVZ9y0m-Naw7GqRUIoWY',
'AVaE7HylNaw7GqRUxTFt']

test2=['AVZ44pI_WoI242VYWZmz',
'AVZ_CpzgI5_rTozncpGe',
'AVZ_CocvI5_rTozncо0D']

test3=['AVaEBVpDIWASxjuiJb8L',
'AVZ48wNuWoI242VYXSMr',
'AVZ95-4BI5_rToznN2cN',
'AVZ4tvQgWoI242VYUEm4']

for a in test1:
    testSet.append([a,"1"])
for a in test2:
    testSet.append([a,"2"])
for a in test3:
```

```
        testSet.append([a,"3"])

testMaster=[]



def getSamples(fname):
        with open(fname) as f:
                content = f.read().splitlines()

        entries=[]
        for line in content:
                aaaa=line.split('[')[1].split(']')[0].split(',')
                meta=aaaa[0:5]
                values=aaaa[5:]
                entries.append([meta,arraystr2num(values)])
        del content
        return entries

def calcTransform(entries):
        out=[]
        out2=[]
        for entry in entries:
                tmp=entry[1]
                total=entry[1][0]
                for i in range(1,31):
                        tmp[i]=float(tmp[i])/float(total)
                entry.append(tmp)
                entry[2].pop(0)
                out.append(entry[2])
                out2.append([entry[0][0], entry[2], entry[0][1]])
                sortToLists(entry[0][0], entry[2], entry[0][1])
        del entries
        pcainner=[]
        for a in range(0,31):
                 pcainner.append(calcPCA(out, a))
        del out
        print len(pcainner)
        return [out2, pcainner]



[out2, pca] = calcTransform(getSamples(fname))


classifications=[]

if runtype == 1:
        transformTrainingSet(pca[24], elMaster)
        for a in out2:
                tmp =closestGroup(pca[24], a)
                if tmp:
                        print tmp
        exit()
del out2

if runtype == 2:
        transformTrainingSet(pca[24], elMaster)
        results=[]
        for a in testMaster:
                tmp=closestGroup(pca[24], a[1])
```

```
                if tmp:
                        results.append([a[1][0],a[0],int(tmp[0]),tmp[1]])

        print results
        score=[]
        score.append([24,len(results)])
        summary=[0,0,0,0]
        for i in results:
                if int(i[1]) == 1:
                        if int(i[2]) == 1:
                                summary[0]=summary[0]+1
                        else:
                                summary[1]=summary[1]+1
                else:
                        if int(i[2]) != 1:
                                summary[2]=summary[2]+1
                        else:
                                summary[3]=summary[3]+1
        score.append(summary)
        print score

if runtype == 3:
        fullscore=[]
        for y in range(0,31):
                tlMaster=[]
                transformTrainingSet(pca[y], elMaster)
                results=[]
                for a in testMaster:
                        tmp=closestGroup(pca[y], a[1])
                        if tmp:
                                results.append([a[1][0],a[0],int(tmp[0]),tmp[1]])

                score=[]
                score.append([y,len(results)])
                summary=[0,0,0,0]
                for i in results:
                        if int(i[1]) == 1:
                                if int(i[2]) == 1:
                                        summary[0]=summary[0]+1
                                else:
                                        summary[1]=summary[1]+1
                        else:
                                if int(i[2]) != 1:
                                        summary[2]=summary[2]+1
                                else:
                                        summary[3]=summary[3]+1
                score.append(summary)
                fullscore.append(score)
        for a in pca[30].explained_variance_ratio_:
                print a
        print fullscore


if runtype == 13:

        [tmpttt, pcattt] = calcTransform(getSamples(fname2))
        del tmpttt
        del pcattt


       fullscore=[]
       for y in range(0,31):
```

```
            tlMaster=[]
            transformTrainingSet(pca[y], elMaster)
            results=[]
            for a in testMaster:
                  tmp=closestGroup(pca[y], a[1])
                  if tmp:
                        results.append([a[1][0],a[0],int(tmp[0]),tmp[1]])

            score=[]
            score.append([y,len(results)])
            summary=[0,0,0,0]
            for i in results:
                  if int(i[1]) == 1:
                        if int(i[2]) == 1:
                              summary[0]=summary[0]+1
                        else:
                              summary[1]=summary[1]+1
                  else:
                        if int(i[2]) != 1:
                              summary[2]=summary[2]+1
                        else:
                              summary[3]=summary[3]+1
            score.append(summary)
            fullscore.append(score)
      for a in pca[30].explained_variance_ratio_:
            print a
      print fullscore



exit()
```