8-4-2017

anomalyDetection: Implementation of Augmented Network Log Anomaly Detection Procedures

Robert J. Gutierrez

Bradley C. Boehmke

Kenneth W. Bauer
Air Force Institute of Technology

Cade M. Saie

Trevor J. Bihl

Recommended Citation

Gutierrez, R. J., Boehmke, B. C., Bauer, K. W., Saie, C. M., & Bihl, T. J. (2017). anomalyDetection: Implementation of augmented network log anomaly detection procedures. R Journal, 9(2), 354–365. https://doi.org/10.32614/RJ-2017-039

anomalyDetection: Implementation of Augmented Network Log Anomaly Detection Procedures

by Robert J. Gutierrez, Bradley C. Boehmke, Kenneth W. Bauer, Cade M. Saie, Trevor J. Bihl

Abstract As the number of cyber-attacks continues to grow on a daily basis, so does the delay in threat detection. For instance, in 2015, the Office of Personnel Management discovered that approximately 21.5 million individual records of Federal employees and contractors had been stolen. On average, the time between an attack and its discovery is more than 200 days. In the case of the OPM breach, the attack had been going on for almost a year. Currently, cyber analysts inspect numerous potential incidents on a daily basis, but have neither the time nor the resources available to perform such a task. **anomalyDetection** aims to curtail the time frame in which anomalous cyber activities go unnoticed and to aid in the efficient discovery of these anomalous transactions among the millions of daily logged events by i) providing an efficient means for pre-processing and aggregating cyber data for analysis by employing a tabular vector transformation and handling multicollinearity concerns; ii) offering numerous built-in multivariate statistical functions such as Mahalanobis distance, factor analysis, principal components analysis to identify anomalous activity, iii) incorporating the pipe operator (`%>%`) to allow it to work well in the **tidyverse** workflow. Combined, **anomalyDetection** offers cyber analysts an efficient and simplified approach to break up network events into time-segment blocks and identify periods associated with suspected anomalies for further evaluation.

Introduction

Organizations worldwide rely heavily on the systems of cyberspace, and the wider Internet as a whole, for commerce, defense operations, infrastructure control systems, financial management, transportation, and other critical services. Unfortunately, the number of cyber-attacks are growing on a daily basis and the ability of organizations to spot anomalous cyber activitiy is becoming more and more delayed (Gutierrez et al., 2017). On average, the time between an attack and its discovery is more than 200 days (Koerner, 2016). In the case of the 2015 Office of Personnel Management breach, in which approximately 21.5 million individual records of Federal employees and contractors had been stolen, the attack had been going on for almost a year prior to the anomalous activity being identified (U.S. Office of Personnel Management, 2015).

Cyber analysts inspect numerous potential incidents on a daily basis, but lack the time and resources to scour the high volumes of data in an efficient manner to identify anomalous activity worth further investigation (Samuelson, 2016). Firewalls and intrusion detection and prevention systems (IDPS) are one line of defense in identifying and stopping suspicious internet traffic. When a suspicious event occurs, these devices generate a log file containing details of what preprogrammed rules were violated and how it was handled (Goodall et al., 2009). Such log files contain details of the event, (i.e. source and destination IP addresses, port numbers, and protocols), but not the packet and data that led to the event. A primary activity of cyber analysts is the analysis of these log files to detect anomalies (Gutierrez et al., 2017). Although a reduced form of anomalous cyber data, these data sets can still represent millions of cyberspace transactions per minute (Jayathilake, 2012). Unfortunately, analysis of these log files has, historically, been heavily manual in nature and often leverages subject matter expertise to find possible threats in logged events to further investigate (Goodall et al., 2009; Jayathilake, 2012; Samuelson, 2016). This approach is inefficient and can suffer from biased, subjective assessments (Zamani, 2013). Moreover, much of the related research has focused on anomaly detection at the device/software level (i.e. Lazarevic et al., 2003; Denning, 1987; Garcia-Teodoro et al., 2009), with little exploration into anomaly detection in the log files generated from the preexisting devices or software (i.e McDonald et al., 2012; Winding et al., 2006; Breier and Branišová, 2015). Consequently, efficient analytic approaches are desirable to help detect anomalous activity in cyber network log data (Gutierrez et al., 2017).

This research introduces the **anomalyDetection** package (Boehmke and Gutierrez, 2017) to provide cyber analysts efficient means for performing anomaly detection in log files. The purpose of the package is to make *identifying* abnormal activity more efficient so that cyber analysts can spend more time researching the potential threat. It is important to note that there is no guarantee that anomalous activity is evidence of malicious cyber activity; however, identification of anomalous activity provides cyber security experts a starting point in their search for undetected malicious activity. **anomalyDetection** simplifies this process.

This paper proceeds as follows. First, we introduce the **anomalyDetection** functions and the

security_logs data set that will be leveraged for the illustrative examples. Next, we demonstrate how **anomalyDetection** can pre-process a data set for follow on analysis. This includes converting a data set with non-numeric attributes into numeric data using a tabulated state vector approach. **anomalyDetection** further prepares this tabulated state vector by inspecting and correcting for multicollinearity. We then illustrate how **anomalyDetection** provides efficient multivariate statistical analysis processes to help identify anomalous activity. Last, we end with some concluding remarks.

anomalyDetection functions

anomalyDetection provides 13 functions to aid in the detection of potential cyber anomalies, which are listed in Table 1. The package also incorporates the pipe operator (%>%) from the **magrittr** package (Bache and Wickham, 2014) for streamlining function composition. To illustrate the functionality of **anomalyDetection** we will use the security_logs data that mimics common information that appears in security logs and comes with **anomalyDetection**. Note that we also load the **tidyverse** package (Wickham, 2017) for common manipulation and visualization tasks.

Table 1: anomalyDetection Functions

Function	Purpose
tabulate_state_vector()	Employs a tabulated vector approach to transform security log data into unique counts of data attributes based on time blocks.
block_inspect()	Creates a list where the original data has been divided into blocks denoted in the state vector.
mc_adjust()	Handles issues with multicollinearity.
mahalanobis_distance()	Calculates the distance between the elements in data and the mean vector of the data for outlier detection.
bd_row()	Indicates which variables in data are driving the Mahalanobis distance for a specific row, relative to the mean vector of the data.
horns_curve()	Computes Horn's Parallel Analysis to determine the factors to retain within a factor analysis.
factor_analysis()	Reduces the structure of the data by relating the correlation between variables to a set of factors, using the eigen-decomposition of the correlation matrix.
factor_analysis_results()	Provides easy access to factor analysis results.
kaisers_index()	Computes scores designed to assess the quality of a factor analysis solution. It measures the tendency towards unifactoriality for both a given row and the entire matrix as a whole.
principal_components()	Relates the data to a set of a components through the eigen-decomposition of the correlation matrix, where each component explains some variance of the data.
principal_components_results()	Provides easy access to principal component analysis results.
get_all_factors()	finds all factor pairs for a given integer.

```
library(tidyverse)
library(anomalyDetection)

security_logs
# A tibble: 300 × 10
#   Device_Vendor Device_Product Device_Action          Src_IP
#           <chr>          <chr>         <chr>           <chr>
# 1         McAfee            NSP       Attempt   223.70.128.61
# 2          CISCO            ASA       Failure 174.110.206.174
# 3            IBM          SNIPS       Success 174.110.206.174
# 4         McAfee            NSP       Success   227.12.127.87
# 5        Juniper            SRX       Success     28.9.24.154
# 6         McAfee            NSP       Success     28.9.24.154
# 7         McAfee            NSP       Attempt     28.9.24.154
# 8         McAfee            ePO       Attempt   223.70.128.61
# 9         McAfee            ePO       Attempt 174.110.206.174
```

```
# 10         CISCO            ASA       Attempt   227.12.127.87
# ... with 290 more rows, and 6 more variables: Dst_IP <chr>,
#   Src_Port <int>, Dst_Port <int>, Protocol <chr>, Country_Src <chr>,
#   Bytes_TRF <int>
```

Data pre-processing

In order to develop a statistical framework for firewall log analysis, data pre-processing is necessary prior to applying any multivariate analytic techniques. To assist in this process **anomalyDetection** offers two main approaches to pre-process log file data - aggregating the data into a tabulated state vector and managing multicollinearity concerns.

First, we can employ the tabulated vector approach introduced by Gutierrez et al. (2017). This approach transforms the security log data into unique counts of data attributes based on pre-defined time blocks. Therefore, as each time block is generated, the categorical fields are separated by their levels and a count of occurrences for each level are recorded into a vector. All numerical fields, such as bytes in and bytes out, are recorded as a summation within the time block. The result is what we call a *state vector matrix*.

Thus, for our `security_logs` data we can create our state vector matrix based on our data being divided into 10 time blocks. What results is the summary of instances for each categorical level in our data for each time block. Consequently, row one represents the first time block and there were 2 instances of CISCO as the device vendor, 1 instance of IBM, etc. By adjusting the `block_length`, `level_limit`, and `level_keep` arguments, the user can refine the level of aggregation and variables to retain and analyze.

```
tabulate_state_vector(security_logs, 10)
# A tibble: 30 × 43
#   CISCO   IBM Juniper McAfee `Palo Alto Networks`   NA1   ASA   ePO
#   <int> <int>   <int>  <int>                <int> <int> <int> <int>
# 1     2     1       1      6                    0     0     2     2
# 2     0     2       4      2                    2     0     0     2
# 3     2     4       2      2                    0     0     2     2
# 4     5     1       2      1                    1     0     5     1
# 5     3     1       1      3                    2     0     3     1
# 6     2     1       2      4                    1     0     2     1
# 7     2     2       1      3                    2     0     2     0
# 8     3     3       1      3                    0     0     3     2
# 9     0     1       4      3                    2     0     0     1
# 10    2     2       2      4                    0     0     2     3
# ... with 20 more rows, and 35 more variables: Firewall <int>, NSP <int>,
#   SNIPS <int>, SRX <int>, NA2 <int>, Attempt <int>, Failure <int>,
#   Success <int>, NA3 <int>, `174.110.206.174` <int>,
#   `223.70.128.61` <int>, `227.12.127.87` <int>, `28.9.24.154` <int>,
#   `89.130.69.91` <int>, NA4 <int>, `145.114.4.203` <int>,
#   `151.194.233.198` <int>, `219.142.109.8` <int>, `32.73.26.223` <int>,
#   `56.137.121.203` <int>, NA5 <int>, TCP <int>, UDP <int>, NA6 <int>,
#   China <int>, India <int>, Korea <int>, Netherlands <int>,
#   Russia <int>, `United Kingdom` <int>, US <int>, NA7 <int>,
#   Src_Port <int>, Dst_Port <int>, Bytes_TRF <int>
```

The state vector matrix provides us with a numerical construct to analyze our log file data; however, prior to proceeding with any multivariate statistical analyses we should inspect the state vector for multicollinearity, to avoid issues such as matrix singularity, rank deficiency, and strong correlation values, and remove any columns that pose an issue. We can use `mc_adjust()` to handle issues with multicollinearity by first removing any columns whose variance is close to or less than a minimum level of variance (`min_var`). Then, it removes linearly dependent columns. Finally, it removes any columns that have a high absolute correlation value equal to or greater than that defined by the user (`max_cor`).

```
(state_vec <- security_logs %>%
  tabulate_state_vector(10) %>%
  mc_adjust())
# A tibble: 30 × 26
#   CISCO   IBM Juniper McAfee `Palo Alto Networks`   ePO Attempt Failure
#   <int> <int>   <int>  <int>                <int> <int>   <int>   <int>
```

```
# 1      2     1       1      6                    0     2       5       1
# 2      0     2       4      2                    2     2       4       2
# 3      2     4       2      2                    0     2       3       3
# 4      5     1       2      1                    1     1       0       5
# 5      3     1       1      3                    2     1       3       3
# 6      2     1       2      4                    1     1       4       1
# 7      2     2       1      3                    2     0       3       3
# 8      3     3       1      3                    0     2       6       2
# 9      0     1       4      3                    2     1       4       4
# 10     2     2       2      4                    0     3       3       0
# ... with 20 more rows, and 18 more variables: `174.110.206.174` <int>,
#   `223.70.128.61` <int>, `227.12.127.87` <int>, `28.9.24.154` <int>,
#   `145.114.4.203` <int>, `151.194.233.198` <int>, `219.142.109.8` <int>,
#   `32.73.26.223` <int>, TCP <int>, China <int>, India <int>,
#   Korea <int>, Netherlands <int>, Russia <int>, `United Kingdom` <int>,
#   Src_Port <int>, Dst_Port <int>, Bytes_TRF <int>
```

By default, `mc_adjust()` removes *all* columns that violate the variance, dependency, and correlation thresholds. Alternatively, we can use `action = "select"` as an argument, which provides interactivity where the user can select the variables that violate the correlation threshold that they would like to remove.

Multivariate statistical analyses

With our data adjusted for multicollinearity we can now proceed with multivariate analyses to identify anomalies in our log file. First we'll use the `mahalanobis_distance()` function to compare the distance between each observation by its distance from the data mean, independent of scale. This is computed as

$$MD = \sqrt{(x - \bar{x})C^{-1}(x - \bar{x})} \tag{1}$$

where x is a vector of p observations, $x = (x_1, \ldots, x_p)$, $\bar{x}$ is the mean vector of the data, $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_p)$, and C^{-1} is the inverse data covariance matrix. Here, we include `output = "both"` to return both the Mahalanobis distance and the absolute breakdown distances and `normalize = TRUE` so that we can compare relative magnitudes across our data.

```
state_vec %>%
  mahalanobis_distance("both", normalize = TRUE) %>%
  as_tibble
# A tibble: 30 × 27
#           MD    CISCO_BD      IBM_BD  Juniper_BD   McAfee_BD
#        <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
# 1  0.4548638 0.005536981 0.013005457 0.016020254 0.021822626
# 2  4.3843567 0.024664733 0.005573767 0.044055697 0.008985787
# 3  0.3604934 0.005536981 0.042732214 0.004005063 0.008985787
# 4  0.8456701 0.050839552 0.013005457 0.004005063 0.016687890
# 5  3.3541555 0.020637838 0.013005457 0.016020254 0.001283684
# 6  1.0900635 0.005536981 0.013005457 0.004005063 0.006418419
# 7  0.6769615 0.005536981 0.005573767 0.016020254 0.001283684
# 8  0.6968967 0.020637838 0.024152991 0.016020254 0.001283684
# 9  0.9910771 0.024664733 0.013005457 0.044055697 0.001283684
# 10 5.7822393 0.005536981 0.005573767 0.004005063 0.006418419
# ... with 20 more rows, and 22 more variables: `Palo Alto
#   Networks_BD` <dbl>, ePO_BD <dbl>, Attempt_BD <dbl>, Failure_BD <dbl>,
#   `174.110.206.174_BD` <dbl>, `223.70.128.61_BD` <dbl>,
#   `227.12.127.87_BD` <dbl>, `28.9.24.154_BD` <dbl>,
#   `145.114.4.203_BD` <dbl>, `151.194.233.198_BD` <dbl>,
#   `219.142.109.8_BD` <dbl>, `32.73.26.223_BD` <dbl>, TCP_BD <dbl>,
#   China_BD <dbl>, India_BD <dbl>, Korea_BD <dbl>, Netherlands_BD <dbl>,
#   Russia_BD <dbl>, `United Kingdom_BD` <dbl>, Src_Port_BD <dbl>,
#   Dst_Port_BD <dbl>, Bytes_TRF_BD <dbl>
```

We can use this information in a modified heatmap visualization (Figure 1) to identify outlier values across our security log attributes and time blocks. Brighter columns represent time blocks

deserving greater attention and further investigation. Larger circles represent variables within a time block that have more anomalous activity. Thus, the larger and brighter the dot the more significant the outlier is and the more it deserves attention.

```
state_vec %>%
  mahalanobis_distance("both", normalize = TRUE) %>%
  as_tibble %>%
  dplyr::mutate(Block = 1:n()) %>%
  gather(Variable, BD, -c(MD, Block)) %>%
  ggplot(aes(factor(Block), Variable, color = MD, size = BD)) +
  geom_point()
```

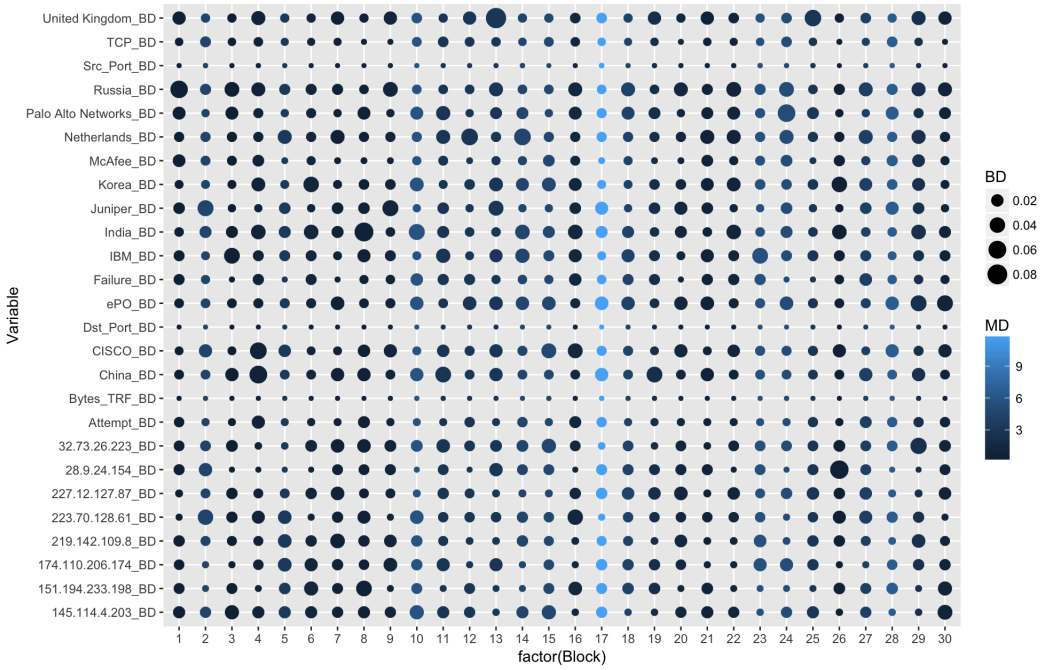

Figure 1: Modified heatmap for time block and feature outlier detection.

We can build onto this with the `bd_row()` function to identify which security log attributes in the data are driving the Mahalanobis distance. `bd_row()` measures the relative contribution of each variable, x_i, to MD by computing

$$BD_i = \left| \frac{x_i - \bar{x}_i}{\sqrt{C_{ii}}} \right| \tag{2}$$

where C_{ii} is the variance of x_i. Furthermore, `bd_row()` will look at a specified row and rank-order the columns by those that are driving the Mahalanobis distance. For example, the plot above identified block 17 as having the largest Mahalanobis distance suggesting some abnormal activity may be occurring during that time block. We can drill down into that block and look at the top 10 security log attributes that are driving the Mahalanobis distance as these may be areas that require further investigation.

```
state_vec %>%
  mahalanobis_distance("bd", normalize = TRUE) %>%
  bd_row(17, 10)
#      Src_Port_BD     Bytes_TRF_BD      Dst_Port_BD  32.73.26.223_BD
#        3.2733887        2.1016995        1.3575754        1.3398650
# 223.70.128.61_BD       McAfee_BD          IBM_BD         Korea_BD
#        1.2376147        1.0828415        1.0372208        0.9979392
#        Russia_BD       Juniper_BD
#        0.9937290        0.8478386
```

Next, we can use factor analysis as a dimensionality reduction technique to identify the underlying structure of the data and identify factors (features) in the data that appear abnormal. Factor analysis

relates the correlations between variables through a set of factors to link together seemingly unrelated variables. The basic factor analysis model is

$$X = \Lambda f + e \tag{3}$$

where X is the vector of responses $X = (x_1, \ldots, x_p)$, f are the common factors $f = (f_1, \ldots, f_q)$, e is the unique factors $e = (e_1, \ldots, e_p)$, and Λ is the factor loadings. Factor loadings are correlations between the factors and the original data and can thus range from -1 to 1, which indicate how much that factor affects each variable. Values close to 0 imply a weak effect on the variable. For the desired results, **anomalyDetection** uses the correlation matrix in its factor analysis computation.

A factor loadings matrix can be computed to understand how each original data variable is related to the resultant factors. This can be computed as

$$\Lambda = \left[\sqrt{\lambda_1} * e_1, \ldots, \sqrt{\lambda_p} * e_p \right] \tag{4}$$

where λ_1 is the eigenvalue for each factor, e_i is the eigenvector for each factor, and p is the number of columns. Factor scores are used to examine the behavior of the observations relative to each factor and can be used to identify anomaly detection. Factor scores are calculated as

$$\hat{f} = X_s R^{-1} \Lambda \tag{5}$$

where X_s is the standardized observations, R^{-1} is the inverse of the correlation matrix, and Λ is the factor loadings matrix. To simplify the results for interpretation, the factor loadings can undergo an orthogonal or oblique rotation. Orthogonal rotations assume independence between the factors while oblique rotations allow the factors to correlate. **anomalyDetection** utilizes the most common rotation option known as varimax. Varimax rotates the factors orthogonally to maximize the variance of the squared factor loadings which forces large factors to increase and small ones to decrease, providing easier interpretation.

To begin using factor analysis, the dimensions of the reduced state vector matrix are first passed to `horns_curve()`, which computes Horn's Parallel Analysis (Horn, 1965) to determine the factors to retain within a factor analysis.

```
horns_curve(state_vec)
# [1] 3.421431145 2.920860390 2.562571534 2.260732869 2.007756392
# [6] 1.789346403 1.595986779 1.418585893 1.255915765 1.106646364
# [11] 0.968567809 0.844067782 0.730563435 0.628690131 0.534405206
# [16] 0.450673920 0.374269067 0.306115502 0.244354685 0.191335630
# [21] 0.144295808 0.103873773 0.070143272 0.043042783 0.022553583
# [26] 0.008092665
```

Next, the dimensionality is determined by finding the eigenvalues of the correlation matrix of the state vector matrix and retaining only those factors whose eigenvalues are greater than or equal to those produced by `horns_curve()`. We use `factor_analysis()` to reduce the state vector matrix into resultant factors. The `factor_analysis()` function generates a list containing five outputs:

`$fa_loadings` numerical matrix with the original factor loadings

`$fa_scores` numerical matrix with the row scores for each factor

`$fa_loadings_rotated` numerical matrix with the varimax rotated factor loadings

`$fa_scores_rotated` numerical matrix with the row scores for each varimax rotated factor

`$num_factors` : numeric vector identifying the number of factors

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  names()
# [1] "fa_loadings"         "fa_scores"           "fa_loadings_rotated"
# [4] "fa_scores_rotated"   "num_factors"
```

For easy access to these results we can use the `factor_analysis_results()` parsing function. The `factor_analysis_results()` function will parse the results either by their list name or by location. For instance to extract the rotated factor scores you can use `factor_analysis_results(data, results = fa_scores_rotated)` or `factor_analysis_results(data, results = 4)` as demonstrated below.

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  factor_analysis_results(4) %>%
  as_tibble
# A tibble: 30 × 11
#             V1         V2         V3          V4          V5         V6
#          <dbl>      <dbl>      <dbl>       <dbl>       <dbl>      <dbl>
# 1  -0.362674625  0.3348386  0.6915550  0.08187007 -0.09327471  1.6577954
# 2  -0.006043119 -0.4959245 -1.7713249  1.48113530  0.75941566  0.4949437
# 3   0.783216952 -0.8394997 -0.4961150  1.31512814  0.97310016 -0.4546721
# 4   0.462460483  1.3648730 -0.1082969 -2.54886990 -0.42566777 -1.9486609
# 5   0.175061099 -0.8789010  1.2417347 -0.36983659  1.55454286 -0.5180766
# 6  -0.207615911  1.2025271  0.2300188  0.40146898  1.23209508  0.3752129
# 7   0.731099082 -1.9734310 -0.9490889 -0.62509695  1.04029889 -0.3571424
# 8   0.030558029 -1.2308883  1.1503857  0.08635927 -1.60839067  1.9569930
# 9   0.265720779 -0.0830922 -1.7467551 -0.13258281  0.47299019 -0.4580762
# 10 -0.875234891 -1.5917696  0.8289798 -1.23316029  1.25799551  0.3656262
# ... with 20 more rows, and 5 more variables: V7 <dbl>, V8 <dbl>,
#   V9 <dbl>, V10 <dbl>, V11 <dbl>
```

To evaluate the quality of a factor analysis solution, Kaiser (Kaiser, 1974) proposed the Index of Factorial Simplicity (IFS). The IFS is computed as

$$IFS = \frac{\sum_i \left[q \sum_s \lambda_{js}^4 - (\sum_s \lambda_{js}^2)^2\right]}{\sum_i \left[(q-1)(\sum_s \lambda_{js}^2)^2\right]} \tag{6}$$

where q is the number of factors, j the row index, s the column index, and λ_{js} is the value in the loadings matrix. Furthermore, Kaiser created the following evaluations of the score produced by the IFS as shown below:

In the .90s Marvelous

In the .80s Meritorious

In the .70s Middling

In the .60s Mediocre

In the .50s Miserable

Less than .50 : Unacceptable

Thus, to assess the quality of our factor analysis results we apply `kaisers_index()` to the rotated factor loadings and, as the results show below, our output value of 0.702 suggests that our results are "middling".

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(data = state_vec, hc_points = .) %>%
  factor_analysis_results(fa_loadings_rotated) %>%
  kaisers_index()
# [1] 0.7018006
```

Furthermore, Figure 2 visualizes the factor analysis results to show the correlation between the columns of the reduced state vector to the rotated factor loadings. Strong negative correlations are depicted as red while strong positive correlations are shown as blue. This helps to identify which factors are correlated with each security log data attribute. Furthermore, this helps to identify two or more security log data attributes that appear to have relationships with their occurrences. For example, this shows that Russia is highly correlated with IP address `223.70.128` since both these attributes are strongly correlated with factor 5. If there is an abnormally large amount of instances with Russian occurrences this would be the logical IP address to start investigating.

```
fa_loadings <- state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  factor_analysis_results(fa_loadings_rotated)

row.names(fa_loadings) <- colnames(state_vec)
```

```
gplots::heatmap.2(fa_loadings, dendrogram = 'both', trace = 'none',
            density.info = 'none', breaks = seq(-1, 1, by = .25),
            col = RColorBrewer::brewer.pal(8, 'RdBu'))
```

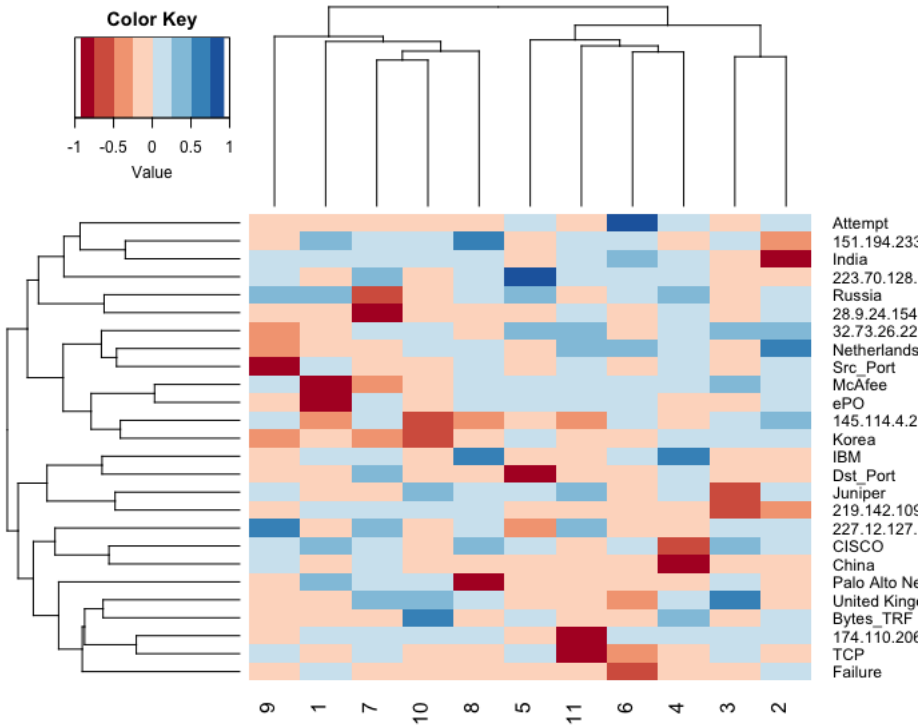

Figure 2: Modified factor analysis heatmap to identify correlated attributes.

We can also visualize the rotated factor score plots as in Figure 3 to see which time blocks appear to be outliers and deserve closer attention.

```
state_vec %>%
  horns_curve() %>%
  factor_analysis(state_vec, hc_points = .) %>%
  factor_analysis_results(fa_scores_rotated) %>%
  as_tibble() %>%
  dplyr::mutate(Block = 1:n()) %>%
  gather(Factor, Score, -Block) %>%
  dplyr::mutate(Absolute_Score = abs(Score)) %>%
  ggplot(aes(Factor, Absolute_Score, label = Block)) +
  geom_text(size = 2) +
  geom_boxplot(outlier.shape = NA)
```

This allows us to look across the factors and identify outlier blocks that may require further intra-block analysis. If we assume that an absolute rotated factor score ≥ 2 represents our outlier cut-off then we see that time blocks 4, 13, 15, 17, 24, 26, and 27 require further investigation. We saw block 17 being highlighted with `mahalanobis_distance()` earlier, but these other time blocks were not as obvious, so by performing and comparing these multiple anomaly detection approaches we can gain greater insights or confirm prior suspicions.

An alternative, yet similar approach to factor analysis is principal component analysis. These two approaches can produce similar outcomes, especially when the error component in equation 4 is close to zero (Fabrigar et al., 1999). However, the results often differ and there are important distinctions in the interpretation of these results (Park et al., 2002). First, a primary difference between the two approaches is that factor analysis estimates errors while principal component analysis does not. This indicates that principal component analysis assumes that the measurement is without error. Second, the goal in factor analysis is to explain the covariances or correlations between the variables. Therefore anomaly detection using factor analysis will identify time blocks and variable attributes in which their

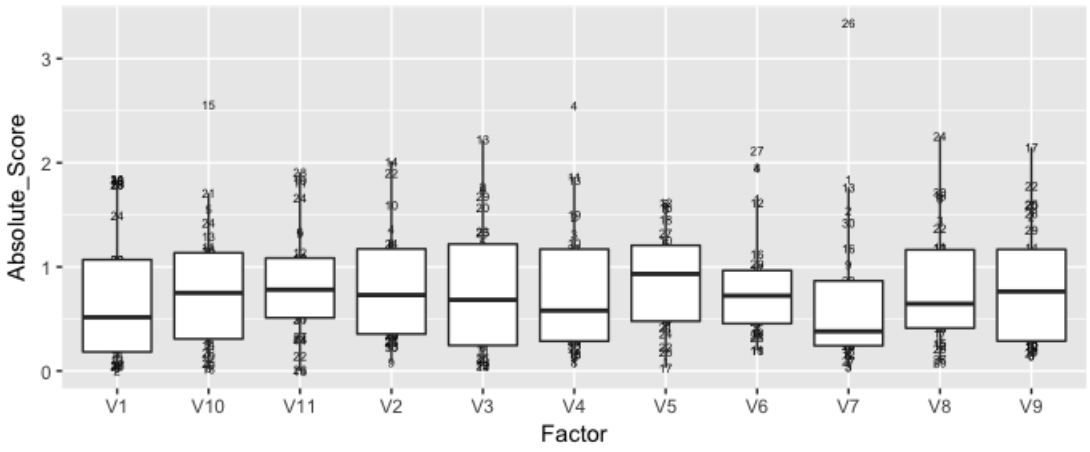

Figure 3: Detecting outlier time blocks based on rotated factor analysis scores.

abnormal behavior is highly correlated to one another. This allows you to identify latent features in the data. By contrast, the goal of principal component analysis is to explain as much of the total variance in the variables as possible. Therefore, if your goal is to reduce the log file variables into a composite component for further analysis, principal component analysis would be appropriate. To maintain clarity between these two approaches the following discussion leverages different notation.

The first principal component of a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p \tag{7}$$

that has the largest variance. By *normalized*, we mean that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. We refer to the elements $\phi_{11}, \ldots, \phi_{p1}$ as the loadings of the first principal component; together, the loadings make up the principal component loading vector, $\phi_1 = (\phi_{11}, \phi_{21}, \ldots, \phi_{p1})^T$. The loadings are constrained so that their sum of squares is equal to one, since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance. After the first principal component Z_1 of the features has been determined, we can find the second principal component Z_2. The second principal component is the linear combination of $X_1, \ldots, X_p$ that has maximal variance out of all linear combinations that are uncorrelated with Z_1. The second principal component scores $z_{12}, z_{22}, \ldots, z_{n2}$ take the form

$$z_{12} = \phi_{12}x_{i1} + \phi_{22}x_{i2} + \cdots + \phi_{p2}x_{ip} \tag{8}$$

where ϕ_2 is the second principal loading vector, with elements $\phi_{12}, \phi_{22}, \ldots, \phi_{p2}$. This continues until all principal components have been computed. Therefore anomaly detection using PCA will maximize the difference in behaviors across time blocks and variable attributes. Thus, identifying anomalies with PCA will identify those attributes that behave very differently than all the other features. To perform a principal components analysis we use `principal_components()` which will create a list containing:

`$pca_sdev` the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix).

`$pca_loadings` the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).

`$pca_rotated` the value of the rotated data (the centered, and scaled if requested, data multiplied by the rotation matrix) is returned.

`$pca_center` the centering used.

`$pca_scale` a logical response indicating whether scaling was used.

```
principal_components(state_vec) %>% names
# [1] "pca_sdev"     "pca_loadings" "pca_rotated"  "pca_center"
# [5] "pca_scale"
```

For easy access to these results we can use the `principal_components_result` parsing function. The `principal_components_result` will parse the results either by their list name or by location. For example, to extract the computed component scores as outlined in Eq. 8 you can use

`principal_components_result(data, results = pca_rotated)` or `principal_components_result(data, results = 3)` as demonstrated below.

```
state_vec %>%
  principal_components() %>%
  principal_components_result(pca_rotated) %>%
  as_tibble
# A tibble: 30 × 26
#          PC1        PC2       PC3        PC4        PC5        PC6
#        <dbl>      <dbl>     <dbl>      <dbl>      <dbl>      <dbl>
# 1    1326.436 -285.25443  36.72628  3.2968478 -0.2977835 -1.3016917
# 2   20404.603  420.02358 236.94988  3.8917064  2.1798436  2.4849683
# 3    1884.370 -229.68499  45.39825  1.2682210  2.4212424  1.2380853
# 4    1555.892  171.11547 116.14809 -6.0588320 -0.5539837 -0.1773544
# 5  -37890.373 -470.91871  55.73837 -0.9419323 -0.3674156 -0.7529821
# 6  -19041.547 -307.40718 -44.48265  0.3985038 -1.2587089 -2.3739508
# 7  -19117.070  130.64729  30.69814 -2.1478778  3.1516592  1.0870282
# 8   21346.072   35.75772  12.44375  1.5784997  2.8835997 -2.4794129
# 9  -18901.619 -251.41102  55.19655  0.1510258 -0.3283588  4.0431385
# 10 -37321.917 -848.69603 -77.71478  2.1905376  2.3692681 -0.3439843
# ... with 20 more rows, and 20 more variables: PC7 <dbl>, PC8 <dbl>,
#   PC9 <dbl>, PC10 <dbl>, PC11 <dbl>, PC12 <dbl>, PC13 <dbl>, PC14 <dbl>,
#   PC15 <dbl>, PC16 <dbl>, PC17 <dbl>, PC18 <dbl>, PC19 <dbl>,
#   PC20 <dbl>, PC21 <dbl>, PC22 <dbl>, PC23 <dbl>, PC24 <dbl>,
#   PC25 <dbl>, PC26 <dbl>
```

We can then follow the principal components analysis with similar visualization activities as performed post-factor analysis to identify features that exhibit abnormal behavior. Since visualizing principal components analysis to identify anomolies mirrors that which we performed in the factor analysis section, we will leave this to the reader as an independent exercise.

Conclusion

Cyber attacks continue to be a growing concern for organizations. Unfortunately, the process of analyzing log files has, historically, been unorganized and lacked efficient approaches. The presented **anomalyDetection** package makes the log file analysis process more efficient and facilitates the identification and analysis of anomalies within log files.

First, **anomalyDetection** improves the pre-processing of cyber data. The package offers functions that help to narrow down abnormal behavior by aggregating internet traffic data into customizable time blocks. Aggregated activity at a higher-level time block should be easier to analyze while still offering a map to suspicious areas to drill down using smaller time blocks. For very large data sets, the analyst can tune the function parameters to start with less blocks and more aggregated data and then iteratively drill down into less aggregated data. Furthermore, **anomalyDetection** improves the process of adjusting for multicollinearity concerns.

Second, **anomalyDetection** improves the modeling process to perform multivariate statistical analysis by offering built-in functions to perform Mahalanobis distance, factor analysis, and principal components analysis along with functions to improve the efficiency of extracting and assessing the results from these multivariate approaches.

Third, we demonstrated how the **anomalyDetection** incorporates the pipe operator (%>%) to allow it to work well in the **tidyverse** workflow, which helps to improve the overall efficiency of the data analysis process.

It is also important to note that although the authors' focus with this package was to target and improve the analysis of network log-file data, **anomalyDetection** can also be used for other large data sets that contain arbitrary features that require data aggregation and anomaly analysis. Furthermore, as with any package, we readily admit that further improvements to the package can be made. Future versions of **anomalyDetection** plan to integrate additional multivariate and time series approaches to offer analysts a wider suite of modeling tools. Integrating plotting functions are also planned for future iterations to further enhance the efficiency of visualizing analytic results. Furthermore, future updates could explore how **anomalyDetection** could interact with distributable systems such as spark by integrating capabilities from packages such as **SparkR**. This would improve **anomalyDetection**'s ability to work with Big data architectures.

Bibliography

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2014. URL https://CRAN.R-project.org/package=magrittr. R package version 1.5. [p355]

B. Boehmke and R. Gutierrez. *anomalyDetection: Implementation of Augmented Network Log Anomaly Detection Procedures*, 2017. URL https://CRAN.R-project.org/package=anomalyDetection. R package version 0.1.1. [p354]

J. Breier and J. Branišová. Anomaly detection from log files using data mining techniques. In *Information Science and Applications*, pages 449–457. Springer, 2015. doi: 10.1007/978-3-662-46578-3_53. URL https://doi.org/10.1007/978-3-662-46578-3_53. [p354]

D. E. Denning. An intrusion-detection model. *IEEE Transactions on software engineering*, (2):222–232, 1987. doi: 10.1109/TSE.1987.232894. URL https://doi.org/10.1109/TSE.1987.232894. [p354]

L. R. Fabrigar, D. T. Wegener, R. C. MacCallum, and E. J. Strahan. Evaluating the use of exploratory factor analysis in psychological research. *Psychological methods*, 4(3):272, 1999. [p361]

P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009. doi: 10.1016/j.cose.2008.08.003. URL https://doi.org/10.1016/j.cose.2008.08.003. [p354]

J. R. Goodall, W. G. Lutters, and A. Komlodi. Developing expertise for network intrusion detection. *Information Technology & People*, 22(2):92–108, 2009. doi: 10.1108/09593840910962186. URL http://dx.doi.org/10.1108/09593840910962186. [p354]

R. Gutierrez, K. Bauer, B. Boehmke, C. Saie, and T. Bihl. Cyber anomaly detection: Using tabulated vectors and embedded analytics for efficient data mining. *Journal of Algorithms and Computational Technology*, forthcoming, 2017. [p354, 356]

J. L. Horn. A rationale and test for the number of factors in factor analysis. *Psychometrika*, 30(2):179–185, 1965. doi: 10.1007/BF02289447. URL https://doi.org/10.1007/BF02289447. [p359]

D. Jayathilake. Towards structured log analysis. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pages 259–264. IEEE, 2012. doi: 10.1109/JCSSE.2012.6261962. URL https://doi.org/10.1109/JCSSE.2012.6261962. [p354]

H. F. Kaiser. An index of factorial simplicity. *Psychometrika*, 39(1):31–36, 1974. doi: 10.1007/BF02291575. URL https://doi.org/10.1007/BF02291575. [p360]

B. I. Koerner. Inside the cyberattack that shocked the US government. *Wired.com*, 2016. URL https://www.wired.com/2016/10/inside-cyberattack-shocked-us-government/. [p354]

A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 25–36. SIAM, 2003. doi: 10.1137/1.9781611972733.3. URL http://dx.doi.org/10.1137/1.9781611972733.3. [p354]

J. T. McDonald, E. D. Trias, M. R. Grimaila, J. Myers, R. F. Mills, and G. Peterson. Design and analysis of a dynamically configured log-based distributed security event detection methodology. *The Journal of Defense Modeling and Simulation*, 9(3):219–241, 2012. doi: 10.1177/1548512911399303. URL https://doi.org/10.1177/1548512911399303. [p354]

H. S. Park, R. Dailey, and D. Lemus. The use of exploratory factor analysis and principal components analysis in communication research. *Human Communication Research*, 28(4):562–577, 2002. [p361]

D. A. Samuelson. Using big data in cybersecurity. *ORMS Today*, 43(5), 2016. [p354]

U.S. Office of Personnel Management. Cybersecurity resource center. Technical report, U.S. Office of Personnel Management, mar 2015. URL https://www.opm.gov/cybersecurity/cybersecurity-incidents. [p354]

H. Wickham. *tidyverse: Easily Install and Load 'Tidyverse' Packages*, 2017. URL https://CRAN.R-project.org/package=tidyverse. R package version 1.1.1. [p355]

R. Winding, T. Wright, and M. Chapple. System anomaly detection: Mining firewall logs. In *Securecomm and Workshops, 2006*, pages 1–5. IEEE, 2006. doi: 10.1109/SECCOMW.2006.359572. URL https://doi.org/10.1109/SECCOMW.2006.359572. [p354]

M. Zamani. Machine learning techniques for intrusion detection. *CoRR*, abs/1312.2177, 2013. URL http://arxiv.org/abs/1312.2177. [p354]

Robert J. Gutierrez
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
rjgutierrez2015@gmail.com

Bradley C. Boehmke
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
bradleyboehmke@gmail.com

Kenneth W. Bauer
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
kenneth.bauer@afit.edu

Cade M. Saie
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
cade.saie@gmail.com

Trevor J. Bihl
Air Force Institute of Technology
2950 Hobson Way
Wright-Patterson AFB, OH 45433
trevor.bihl@gmail.com