Table of Contents

Preliminary Content

Acknowledgements

I thank my advisor, Professor Peter Hoff, and the Director of Undergraduate Studies, Professor Mine Cetinkaya-Rundel, for their guidance in this project. I also thank Duke University's Statistics Department and Office of Information Technology, especially my dataset contact at OIT, Eric Hope, for making this project possible. Most of all I thank my parents for their continued unwavering support in all my endeavors.

The goal of this project is to identify novel methods for detecting anomalies in network IP data. The space is represented as a 3-dimensional tensor of the continuous features (source bytes, destination bytes, source packets, destination packets) divided by their respective source port and destination port combinations. This project implements and assesses the validity of principal component analysis and matrix completion via singular value decomposition (more methods pending) in determining anomalous entries in the tensor.

Chapter 1

Introduction

1.1 Anomaly Detection

Anomaly detection is used to identify unusual patterns or observations that do not conform to expected behavior in a dataset. Anomalies can be broadly categorized into three categories:

Point anomalies: A single instance of data is anomalous if it's too far off from the rest. For example detecting credit card fraud based on a single spending spree that represents the credit card being stolen and used.

Contextual anomalies: The abnormality is context specific. This type of anomaly is common in time-series data. For instance, high spending on food and gifts every day during the holiday season is normal, but may be considered unusual otherwise.

Collective anomalies: A set of data observations that when collectively assessed helps in detecting anomalies. For instance, repeated pings from a certain IP address to a port connection on a hosted network may be classified as a port scanner, which often preludes a network attack.

1.2 Network Attacks

Network security is becoming increasingly relevant as the flow of data, bandwith of transactions, and user dependency on hosted networks increase. As entire networks grow in nodes and complexity, attackers gain easier entry points of access to the network. The most benign of attackers attempt to shutdown networks (e.g. causing a website to shutdown with repeated pings to its server), while more malicious attempts involve hijacking the server to publish the attacker's own content or stealing unsecured data from the server, thus compromising the privacy of the network's users.

Attackers follow a specific three step strategy when gathering intelligence on a network, the most important component of which is scanning. Network scanning is a procedure for identifying active hosts on a network, the attacker uses it to find information about the specific IP addresses that can be accessed over the Internet, their target's operating systems, system architecture, and the services running on each node/computer in the network. Scanning procedures, such as ping sweeps and

port scans, return information about which IP addresses map to live hosts that are active on the Internet and what services they offer. Another scanning method, inverse mapping, returns information about what IP addresses do not map to live hosts; this enables an attacker to make assumptions about viable addresses.

All three of these scanning methods leave digital signatures in the networks they evaluate because they apply specific pings that are then stored in the network logs. Most scanners use a specific combination of bytes, packets, flags (in TCP protocol), and ports in a sequence of pings to a network. Identifying a scanner's often many IP addresses from the set of pings available in the network's logs is thus an anomaly detection problem. In particular, because the data is unlabeled, meaning it is unclear which observations are actually scanners and which are just standard user behavior, unsupervised approaches are necessary for tackling the problem.

This particular dataset is from Duke University's Office of Information Technology (OIT), and it covers all observations in their network traffic during a five minute period in February 2017.

1.2.1 Status Quo Solution

OIT's current solution for detecting scanners relies on specific domain knowledge gathered from diagnostics programs and data analysis completed on previous data. They prevent scanners by blocking IP addresses that fit certain rules they have constructed to run on every network transaction as it occurs. The specific checks in these rules are private for security reasons, but they belong to the nature of evaluating the size of transactions, repeated connections between particular ports, many pings from the same address, and combinations of these particular behaviors.

While this solution presents a methodical way for banning IP addresses and its method of rule checking is essentially removing what OIT considers outliers for network transactions-any observation that does not fit within the constraints specified by the rules is classified as an outleir and its source IP is blocked-it is inflexible, prone to detecting false negatives, and fails to detect observations that may be within the parameter constraints of the rules but are anomalous with respect to other parameters or parameter constraints.

1.3 Network Dataset

1.3.1 Features

The networks dataset contains 13 features, 8 categorical and 5 continuous, and the observations are unlabeled (not specified whether they are considered a scanner). The 13 features are:

Continuous:

- StartTime (Start Time): the time when the observation is logged
- SrcBytes (Source Bytes): the total number of bytes sent in the observation
- SrcPkts (Source Packets): the number of packets sent in the observation

- DstBytes (Destination Bytes): the total number of bytes received in the observation
- DstPkts (Destination Packets): the number of packets received in the observation Note, the destination packets and bytes features do not have the same values as their source counterparts because the connections are compressed and decompressed into different forms and byte sizes when sent. For instance, it is possible for the number of destination packets to be larger than source packets. It is also possible for information to be lost during the connection.

Categorical:

- Flgs (connection flag): flow state flags seen in transaction between the two addresses
- Proto (network protocol): specifies the rules used for information exchange via network addresses. Transmission Control Protocol (TCP) uses a set of rules to exchange messages with other Internet points at the information packet level, and Internet Protocol (IP) uses a set of rules to send and receive messages at the Internet address level.
- SrcAddr (Source Address): the IP address of the connection's source
- DstAddr (Destination Address): the IP address of the connection's destination
- Sport (Source Port): the network port number of the connection's source. A port numbers identifies the specific process to which a network message is forwarded when it arrives at a server.
- Dport (Destination Port): the network port number of the connection's destination
- Dir (direction): the direction of the connection
- State (connection state): a categorical assessment of the current phase in the transaction when the timestamp is recorded

Note, the addresses have been anonymized for security reasons.

1.3.2 Argus

Argus is the open source network security tool applied to network transactions that collects the data for the features. The Argus wiki and the OIT manual provides key insights into the structure and nature of the data. Specifically, the sessions are clustered together by address, so the pytes and packets values are accumulative over a set duration and each session has its own start time but does not have a tracked end time. There exist 2-4 million connections on average every 5 minutes. Furthermore the protocol in this dataset is always gathered from TCP protocol and the direction will always be to the right (i.e. Source to Destination). This information supports dropping proto, StartTime, and Direction from the dataset for future analysis because they do not present any information regarding whether an observation can be considered an anomaly. Furthermore, the State feature may not be reliable because Argus occasionally resets the state data statistics during monitoring.

Chapter 2

Modeling Port Relationships

2.1 Motivation

Exploratory data analysis signaled that there may exist trends between different port combinations. For instance, a particular source and destination port may frequently contain large byte transactions in their connections. Devising a systematic way to identify these combinations may present outliers that can be further investigated for scanner behavior.

Beginning the investigation requires the creation of a 2-dimensional matrix of the most common source and destination port combinations. Each cell in this matrix is of variable length and holds all of the data observations involving that particular source and destination port pairing. Following the creation of this matrix, principal component analysis may be applied to the combinations present in each cell to investigate the behavior of that specific connection.

2.2 Principal Component Analysis

Principal component analysis represents data in terms of its principal components rather than relying on traditional Cartesian axes. Principal components contain the underlying structure in data by representing the directions that contain the most variance. Each successive principal component is orthoganol to the previous, so the resulting vectors yields an uncorrelated orthogonal basis set. Because PCA is sensitive to relative scaling of variables, a normal scores transformation is applied before the algorithm is run.

2.2.1 Application

In this problem, Principal Component Anaylsis is applied to each source-destination port grouping of observed data to understand the underlying structure of the data partition's continuous features: source bytes and packets and destination bytes and packets. In particular the amount of variance explained by the generated principal components and their relative directions will signal whether specific trends in connec-

tion behavior occur at certain ports, and whether the size of each of the continuous features affects port behavior as well as the other features.

2.3 Implementation

2.3.1 Ports Combination Matrix/Tensor

```
#set counts
s_num = 25
d_num = 25
combo_num = 20

#get top sport and dport values
Sport_table = table(Sport)
Sport_table = as.data.frame(Sport_table)
Sport_table = Sport_table[order(-Sport_table$Freq),]
top_Sport = (head(Sport_table$Sport, s_num))

Dport_table = table(Dport)
Dport_table = as.data.frame(Dport_table)
Dport_table = Dport_table[order(-Dport_table$Freq),]
top_Dport = (head(Dport_table$Dport, d_num))

#subset data for the combinations
argus_maxes = argus[is.element(Sport, top_Sport) & is.element(Dport, top_Dport), ]
argus_maxes = transform(argus_maxes,
                        Sport = as.numeric(as.character(Sport)),
                        Dport = as.numeric(as.character(Dport)))
max_combinations = as.data.frame(table(argus_maxes$Sport, argus_maxes$Dport))

top_combinations = head(max_combinations[order(-max_combinations$Freq),], combo_num)
top_combinations$Sport = top_combinations$Var1
top_combinations$Dport = top_combinations$Var2
top_combinations$Var1 = NULL
top_combinations$Var2 = NULL
top_combinations = transform(top_combinations,
                             Sport = as.numeric(as.character(Sport)),
                             Dport = as.numeric(as.character(Dport)))

#generate the combinations matrix of ports
extract_intersection = function(sport, dport){
  argus_subset = argus[Sport == sport & Dport == dport,]
  return (argus_subset)
```

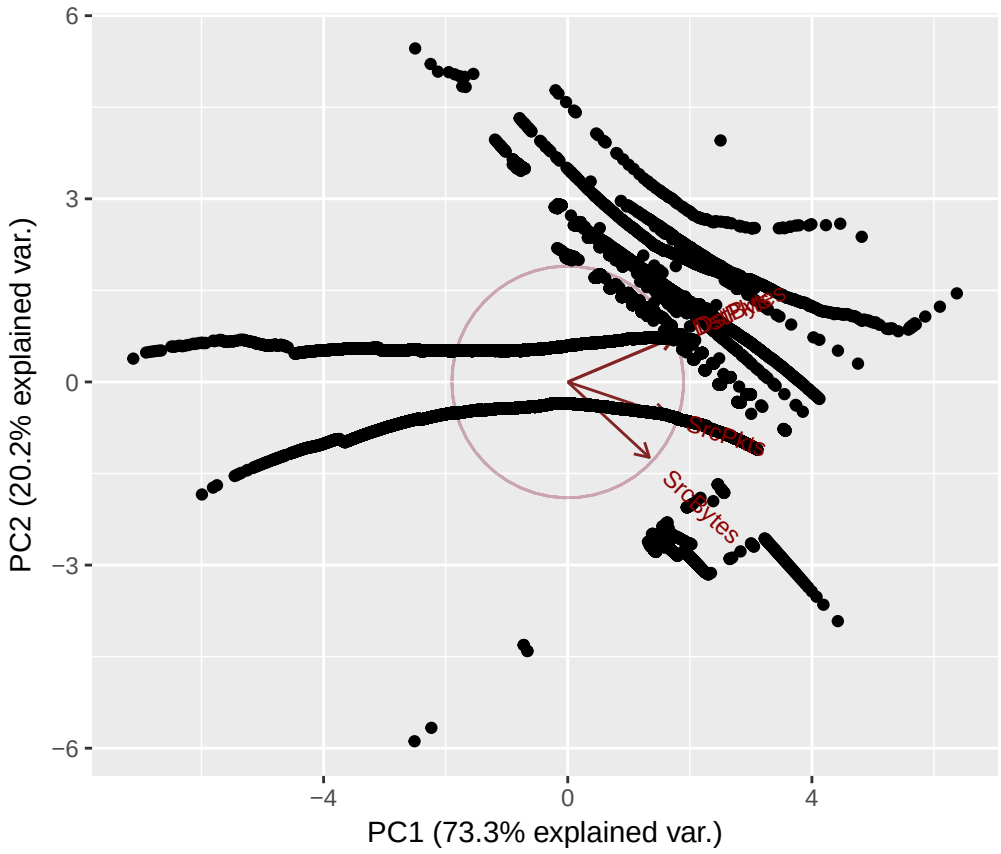
```
}
generate_combinations_matrix = function(top_combinations){
  n = dim(top_combinations)[1]
  combinations = c()
  for (i in 1:n){
    sport = as.numeric(top_combinations[i,]$Sport)
    dport = as.numeric(top_combinations[i,]$Dport)
    combo = extract_intersection(sport, dport)
    combinations = c(combinations, list(combo))
  }
  return (combinations)
}
combinations = generate_combinations_matrix(top_combinations)
```

2.3.2 Investigating Combinations

```
#principal component analysis and visualizing results
pca_analysis = function(SrcBytes, SrcPkts, DstBytes, DstPkts){
  pca_cont_vars = cbind(SrcBytes, SrcPkts, DstBytes, DstPkts)
  pca = prcomp(pca_cont_vars, center = TRUE, scale. = TRUE)
  print(pca$rotation)
  print((summary(pca)))
  #screeplot(pca, type="lines",col=3)
  g = ggbiplot(pca, obs.scale = 1, var.scale = 1,
               ellipse = TRUE,
               circle = TRUE)
  g = g + scale_color_discrete(name = '')
  g = g + theme(legend.direction = 'horizontal',
                legend.position = 'top')
  print(g)
  return(pca$rotation)
}
#apply pca to the data partitions on the top 10 port combinations
combo_num = 10
for (i in 1:combo_num){
  combo_table = combinations[i]
  combo_table = transform(combo_table,
                          SrcBytes = as.numeric(SrcBytes),
                          SrcPkts = as.numeric(SrcPkts),
                          DstBytes = as.numeric(DstBytes),
                          DstPkts = as.numeric(DstPkts))
  cat("Sport:", combo_table$Sport[1],"\t")
  cat("Dport:", combo_table$Dport[1],"\n")
```

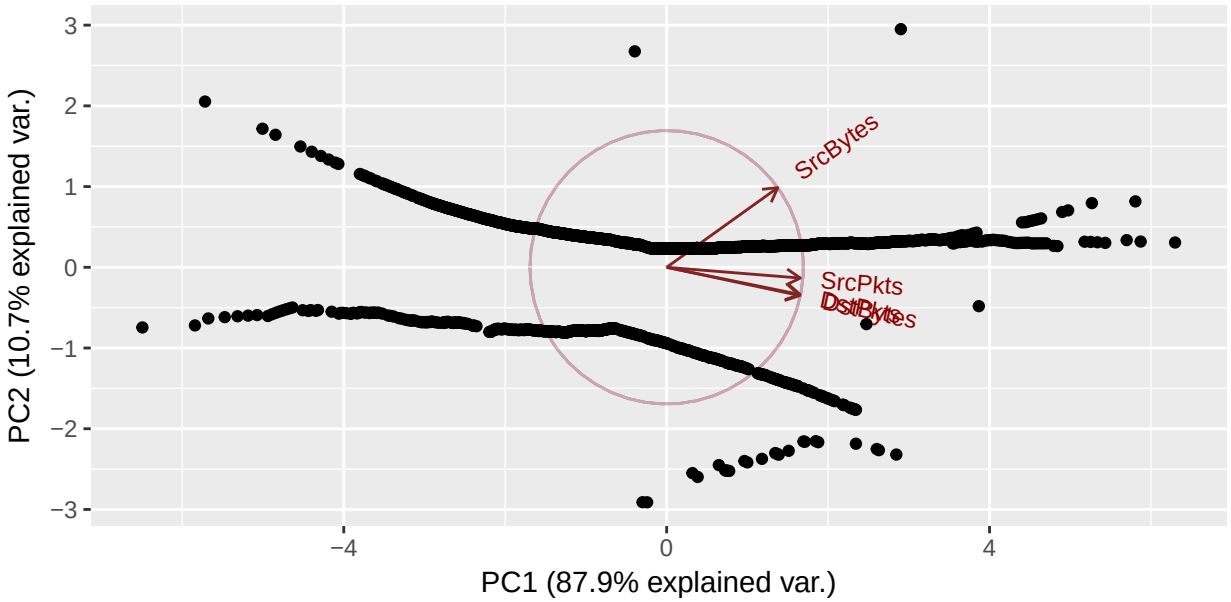
```
  SrcBytes_norm =  nscore(combo_table$SrcBytes)$nscore
  SrcPkts_norm =  nscore(combo_table$SrcPkts)$nscore
  DstBytes_norm =  nscore(combo_table$DstBytes)$nscore
  DstPkts_norm =  nscore(combo_table$DstPkts)$nscore
  pca_analysis(SrcBytes_norm, SrcPkts_norm, DstBytes_norm, DstPkts_norm)
}
```

```
Sport: 32416    Dport: 9163
               PC1        PC2        PC3          PC4
SrcBytes 0.4113308 -0.7246604 -0.5528760  0.001575730
SrcPkts  0.5054262 -0.3234250  0.7999554  0.003459252
DstBytes 0.5357591  0.4327433 -0.1665939  0.705649994
DstPkts  0.5369483  0.4277813 -0.1632360 -0.708550377
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7118 0.8991 0.51065 0.02309
Proportion of Variance 0.7326 0.2021 0.06519 0.00013
Cumulative Proportion  0.7326 0.9347 0.99987 1.00000
```

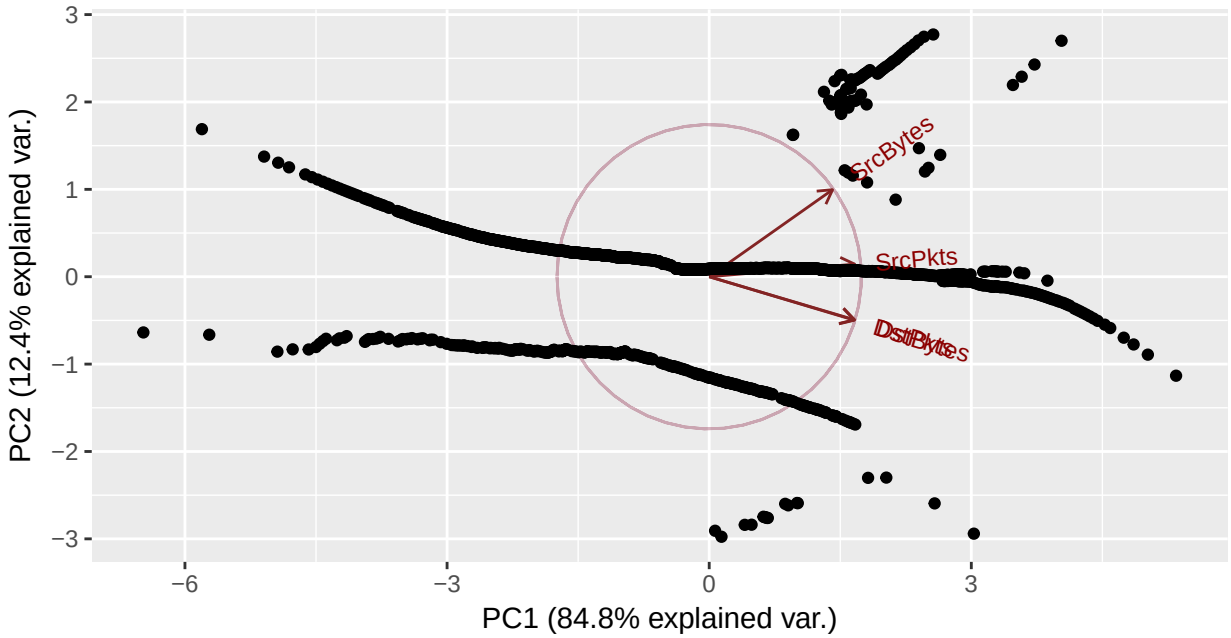



```
Sport: 4145     Dport: 9119
               PC1        PC2        PC3          PC4
SrcBytes 0.4333309  0.8902089 -0.1405308  0.001886624
```

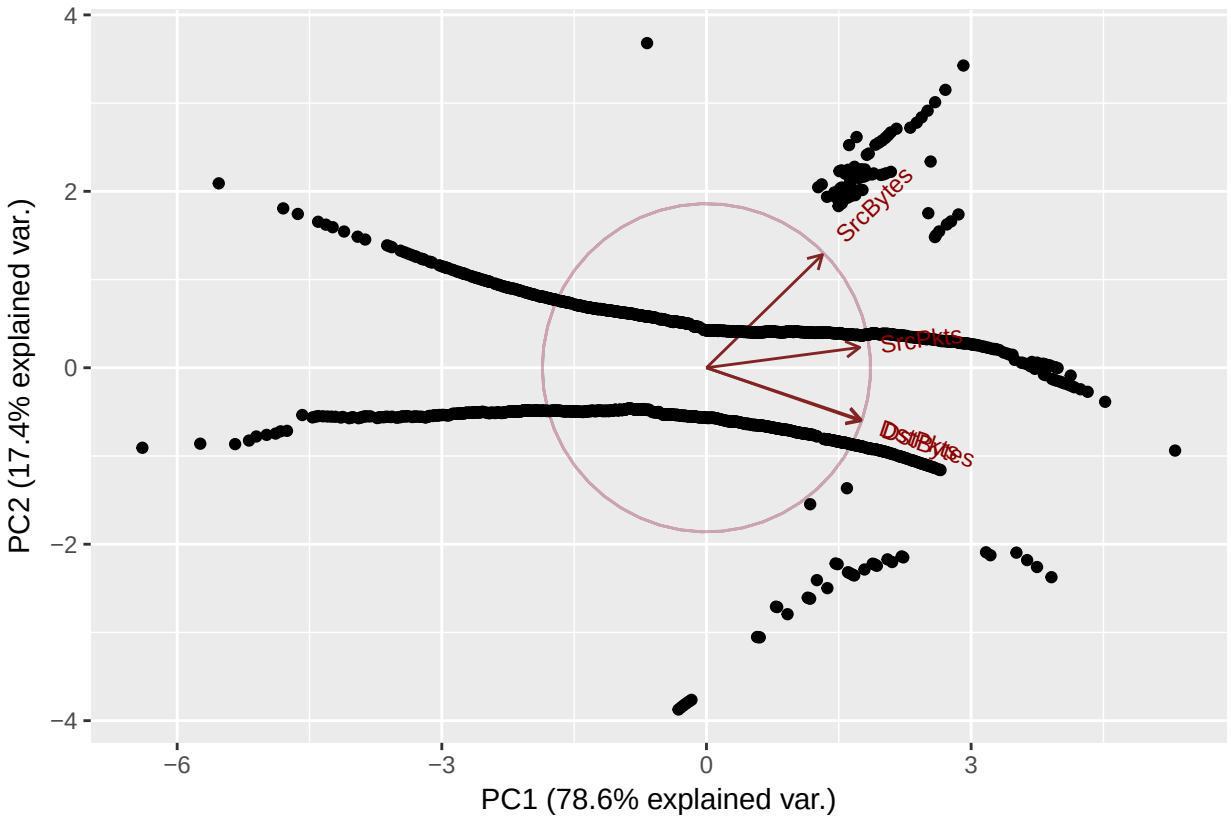
```
SrcPkts  0.5213529 -0.1206221  0.8439979  0.036180840
DstBytes 0.5190742 -0.3165495 -0.3953966  0.688490995
DstPkts  0.5205549 -0.3045896 -0.3340362 -0.724339380
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.875 0.6538 0.23124 0.05551
Proportion of Variance 0.879 0.1069 0.01337 0.00077
Cumulative Proportion  0.879 0.9859 0.99923 1.00000
```



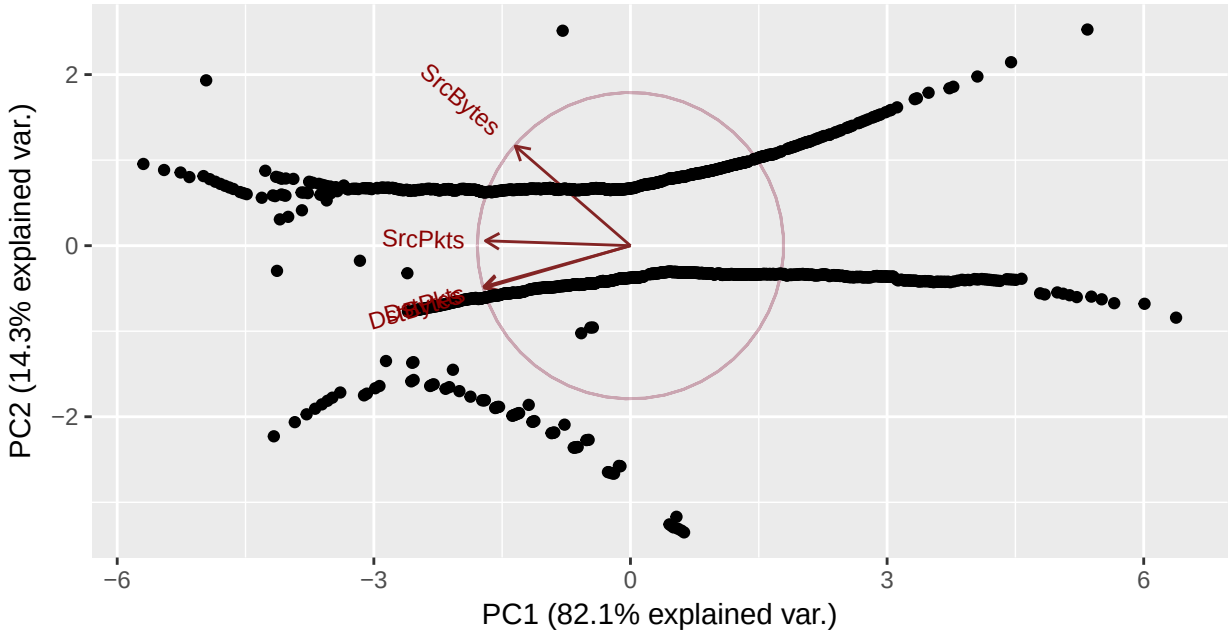

```
Sport: 19239    Dport: 9153
               PC1        PC2        PC3          PC4
SrcBytes 0.4400190  0.8125062 -0.3823817 -0.001104223
SrcPkts  0.5189497  0.1174251  0.8466938 -0.003489750
DstBytes 0.5179403 -0.4057384 -0.2640886 -0.705245607
DstPkts  0.5184712 -0.4017728 -0.2591352  0.708953621
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.8418 0.7037 0.33316 0.03695
Proportion of Variance 0.8481 0.1238 0.02775 0.00034
Cumulative Proportion  0.8481 0.9719 0.99966 1.00000
```



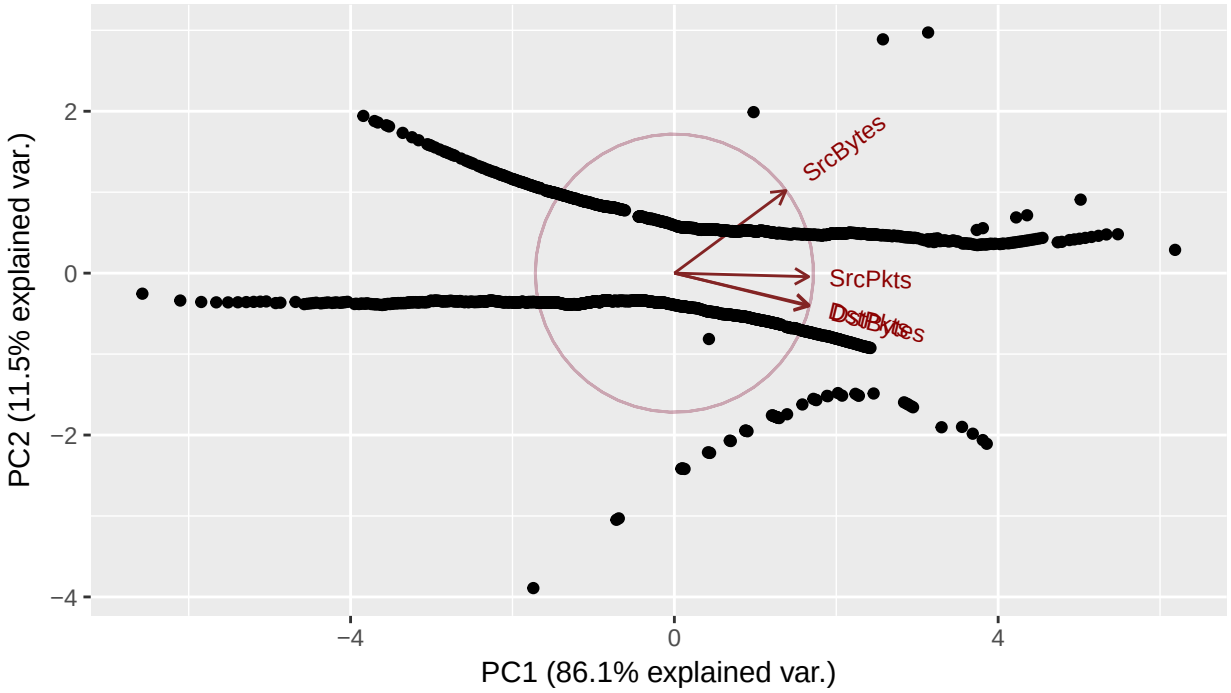

```
Sport: 4243     Dport: 27
               PC1        PC2        PC3          PC4
SrcBytes 0.3986539  0.8262979 -0.3978739  0.001762539
SrcPkts  0.5268775  0.1487193  0.8367993  0.007045781
DstBytes 0.5300998 -0.3876467 -0.2708012  0.703840168
DstPkts  0.5314785 -0.3805842 -0.2610173 -0.710321243
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7727 0.8343 0.40033 0.03406
Proportion of Variance 0.7856 0.1740 0.04007 0.00029
Cumulative Proportion  0.7856 0.9596 0.99971 1.00000
```



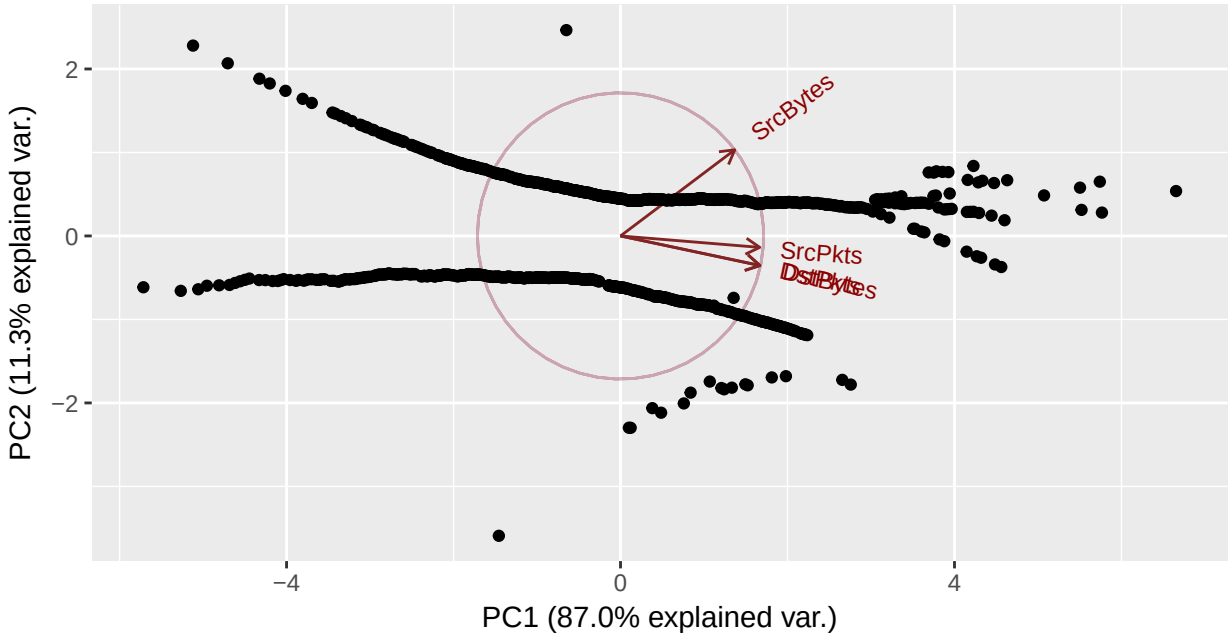

```
Sport: 4243     Dport: 10290
                PC1         PC2        PC3          PC4
SrcBytes -0.4146349  0.86007431 -0.2972268 -0.002501828
SrcPkts  -0.5225073  0.04229265  0.8514240 -0.016572825
DstBytes -0.5257892 -0.36545098 -0.3181251 -0.699133556
DstPkts  -0.5278350 -0.35345310 -0.2924547  0.714794623
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.8125 0.7560 0.37528 0.05022
Proportion of Variance 0.8213 0.1429 0.03521 0.00063
Cumulative Proportion  0.8213 0.9642 0.99937 1.00000
```



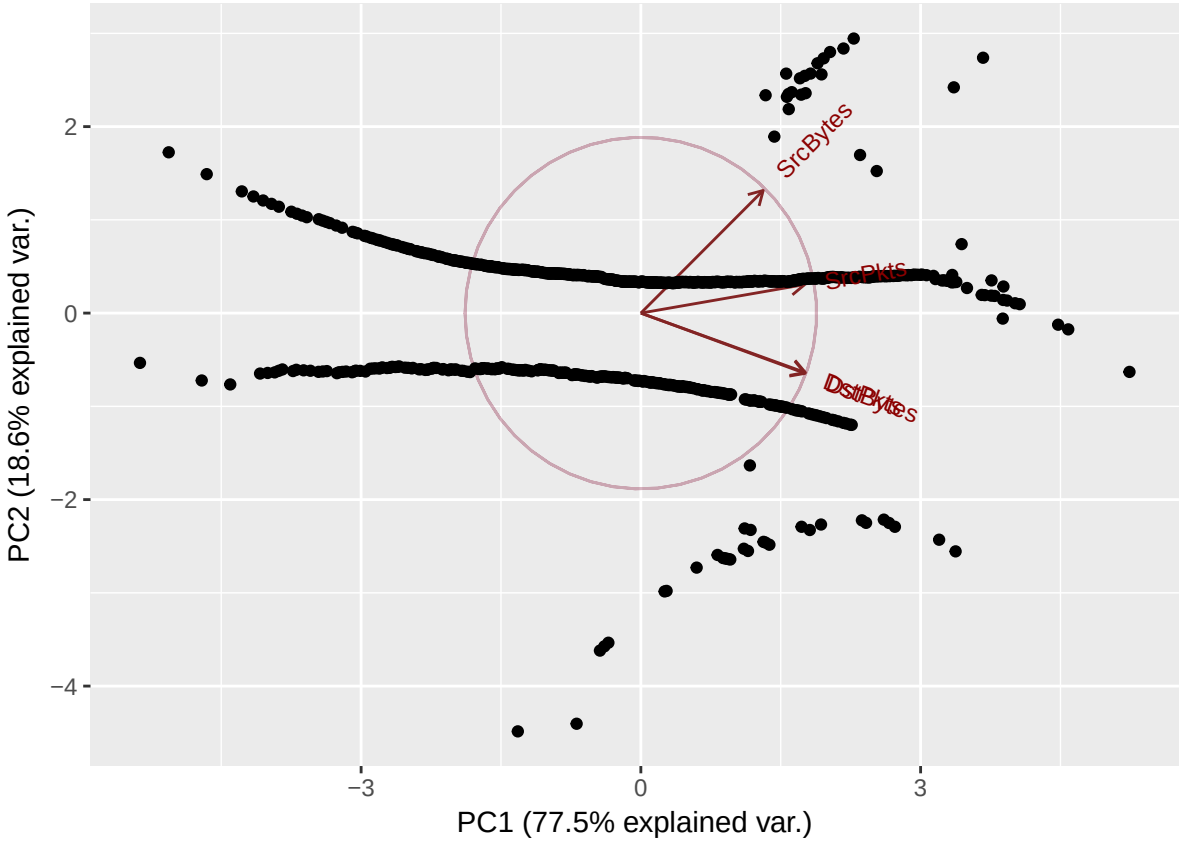

```
Sport: 4243     Dport: 26
               PC1         PC2        PC3          PC4
SrcBytes 0.4320683  0.87332152 -0.2249932  0.002131619
SrcPkts  0.5202875 -0.03769203  0.8529953  0.016712305
DstBytes 0.5202102 -0.34831316 -0.3463831  0.698625865
DstPkts  0.5215355 -0.33847715 -0.3190547 -0.715288791
Importance of components:
                          PC1    PC2     PC3    PC4
Standard deviation     1.8559 0.6797 0.30326 0.0398
Proportion of Variance 0.8611 0.1155 0.02299 0.0004
Cumulative Proportion  0.8611 0.9766 0.99960 1.0000
```



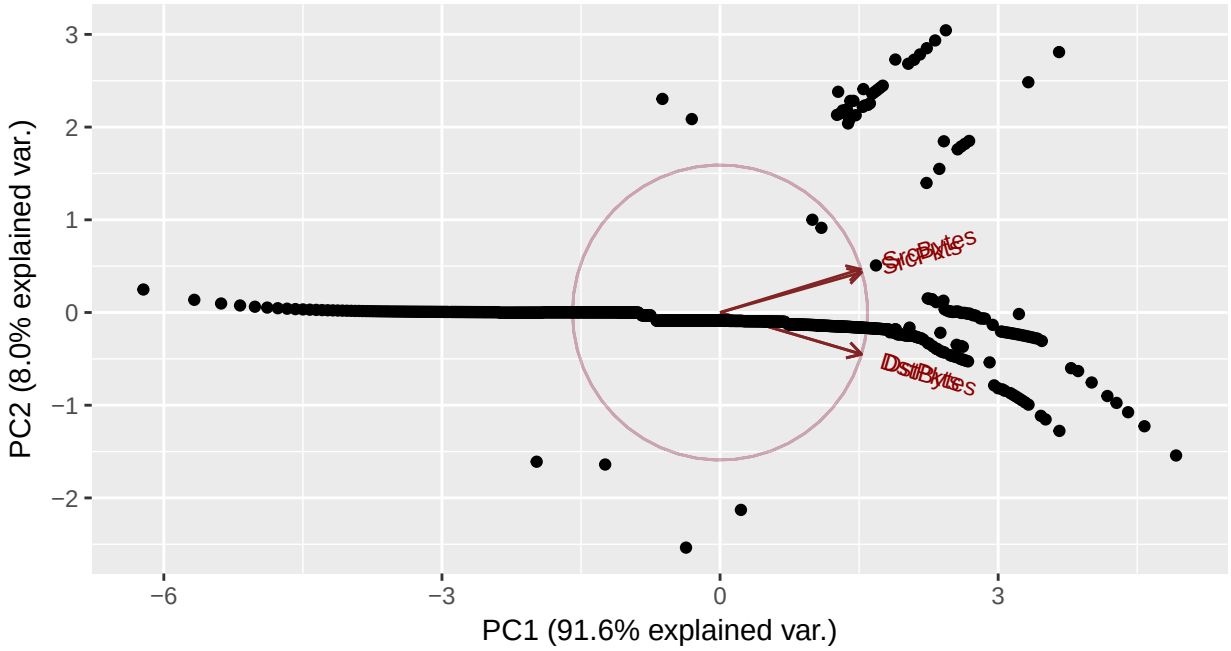

```
Sport: 4243     Dport: 25
               PC1        PC2        PC3           PC4
SrcBytes 0.4278586  0.8931432 -0.1386797 -0.0005117289
SrcPkts  0.5214493 -0.1186010  0.8449812 -0.0055942005
DstBytes 0.5218835 -0.3078511 -0.3699338 -0.7042828114
DstPkts  0.5221736 -0.3057071 -0.3604494  0.7098972916
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.8651 0.6736 0.25860 0.02853
Proportion of Variance 0.8697 0.1134 0.01672 0.00020
Cumulative Proportion  0.8697 0.9831 0.99980 1.00000
```



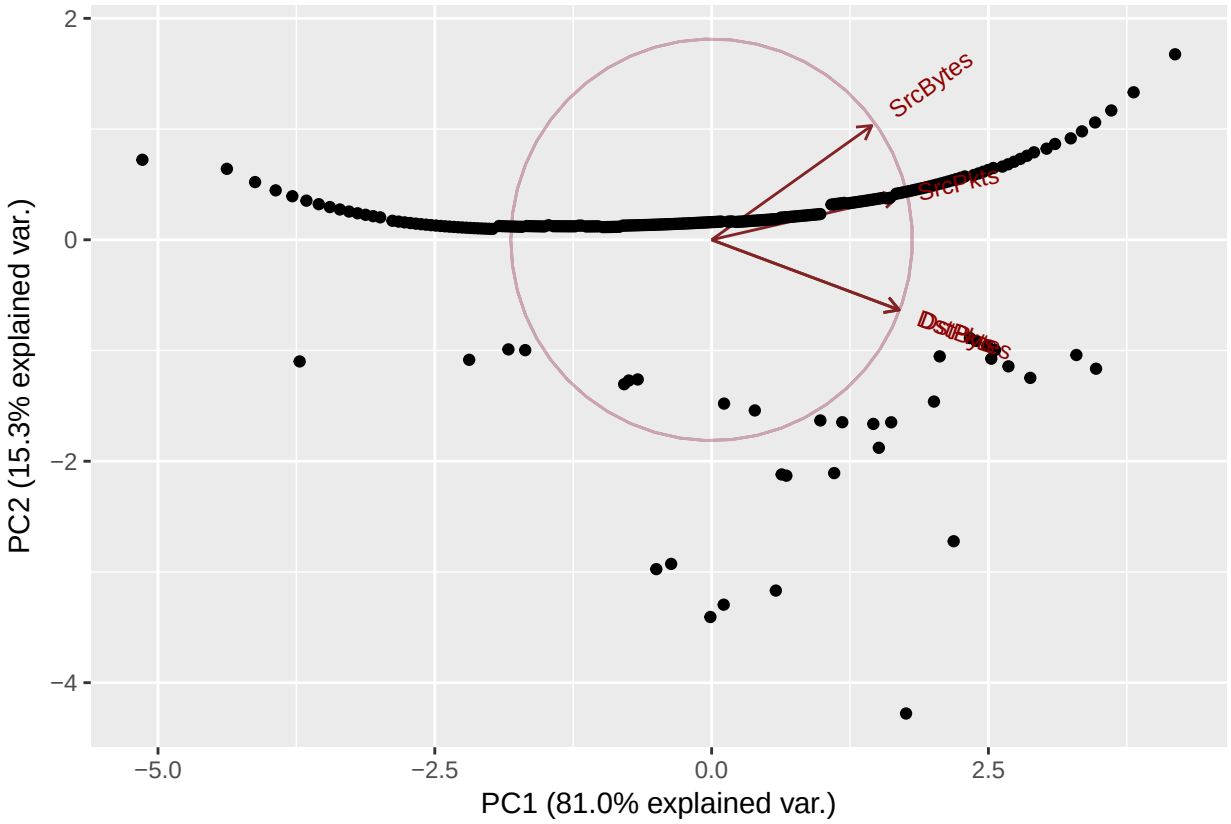

```
Sport: 4243     Dport: 10282
               PC1        PC2        PC3          PC4
SrcBytes 0.3952127  0.8082469 -0.4365116  0.001226653
SrcPkts  0.5292182  0.1880550  0.8273666  0.005281099
DstBytes 0.5303470 -0.3972687 -0.2534339  0.704755906
DstPkts  0.5314763 -0.3918544 -0.2463602 -0.709429149
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7608 0.8629 0.39223 0.03300
Proportion of Variance 0.7751 0.1862 0.03846 0.00027
Cumulative Proportion  0.7751 0.9613 0.99973 1.00000
```




```
Sport: 19581    Dport: 118
               PC1        PC2         PC3           PC4
SrcBytes 0.4978768  0.5190781 -0.69474927  0.0003069068
SrcPkts  0.5010460  0.4817043  0.71896568 -0.0014778396
DstBytes 0.5004716 -0.4999998 -0.01523225 -0.7066090712
DstPkts  0.5005994 -0.4985169 -0.01340796  0.7076025312
Importance of components:
                          PC1     PC2     PC3     PC4
Standard deviation     1.9145 0.56579 0.10402 0.06249
Proportion of Variance 0.9163 0.08003 0.00271 0.00098
Cumulative Proportion  0.9163 0.99632 0.99902 1.00000
```




```
Sport: 35506    Dport: 12
               PC1        PC2        PC3         PC4
SrcBytes 0.4432713  0.7265363  0.5248198 -0.01482290
SrcPkts  0.5148990  0.2730712 -0.8122575  0.02343025
DstBytes 0.5185367 -0.4458439  0.1990667  0.70193688
DstPkts  0.5191429 -0.4458702  0.1586646 -0.71169932
Importance of components:
                         PC1    PC2     PC3     PC4
Standard deviation     1.80 0.7830 0.38006 0.05046
Proportion of Variance 0.81 0.1533 0.03611 0.00064
Cumulative Proportion  0.81 0.9633 0.99936 1.00000
```



2.3.3 Interpretation

In general the first two principal components explained most of the variance (~90%) for each of the port combinations. The scatterplots of the principal components show clear horizontal patterns in the 2nd principal component. This similar behavior, mirrored throughout the top 10 most frequent ports, may be caused by the high frequency of zeroes in the dataset. Recall, the first quartile of observations for destination bytes and packets were all 0. This high frequency of the same value (0) yields ties when performing the normal scores transformation applied to the data, which may also cause the horizontal behavior exhibited in every principal component analysis.

Chapter 3

Tensor Completion

3.1 Motivation

This approach to the anomaly detection problem reduces the dataset to the values of the four continuous features, SrcBytes, SrcPkts, DstBytes, DstPkts, observed across different source port and destination port combinations. The data can be represented as a 3-dimensional tensor $Y \in \mathbb{R}_{\geqslant \times \leqslant \times \not\geqslant}$ where m represents the number of source ports, n represents the number of destination ports, and 4 accounts for the four continuous features in the dataset. Each cell, y_{ijk}, contains the mean of all the observations observed for source port, i, and destination port j. In the cases where the combination of i and j is not observed in the dataset, y_{ijk} is missing.

The goal of this project is to devise an optimal strategy for imputing the missing cells in Y to create the completed tensor $Y' \in \mathbb{R}_{\geqslant \times \leqslant \times \not\geqslant}$. As new observations are observed for combinations of ports i and j, the y'_{ijk} values can be interpreted as an approximation for the expected behavior for that particular port combination. Observations with continuous features that are a certain threshold away from y'_{ijk} may be marked as anomalies and investigated further.

3.2 Imputation Strategy

The imputation strategy focuses on finding a low rank approximation for Y when decomposing the tensor.

3.2.1 CP Decomposition

The CP decomposition expresses the tensor as:

$$Y = \sum_{r=1}^{R} u_r \cdot v_r \cdot w_r$$

where r represents the rank approximation, $\cdot$ denotes the outer product of tensors, and $u \in \mathbb{R}_{\diagdown\times\diagdown}$, $v \in \mathbb{R}_{\diagdown\times\diagdown}$, and $w \in \mathbb{R}_{\diagdown\times\diagdown}$. Each individual cell is expressed:

$$y_{ijk} = \sum_{r=1}^{R} u_{ri} \cdot v_{ri} \cdot w_{ri}$$

Applying this decomposition yields the objective

$$min_{Y'} \|Y - Y'\|, Y' = \sum_{r=1}^{R} \lambda_r (u_r \cdot v_r \cdot w_r)$$

, where $lambda_r$ is the regularization penalty.

3.2.2 Variable Sample Sizes

A traditional approach to tensor completion involves using alternating least squares regression to impute the missing values after populating them with some initial values. The next section applies this approach to a 2-dimensional $m \times n$ tensor that represents a cross-section slice of Y that only includes one of the four continuous features. *include als section here, related work:* `https://arxiv.org/abs/1410.2596` *(hastie fast als), application netflix challenge*

While this approach yields a completed tensor, it does not account for the fact that the means in each cell are calculated from a variable number of observations. Furthermore it is not necessarily true that $n_{ijk} = n_{i'j'k'}$ or $\sigma^2_{ijk} = \sigma^2_{i'j'k'}$ for $i \neq i', j \neq j', k \neq k'$.

We propose the following model:

$$y_{ijk} \sim N(\mu_{ijk}, \frac{\sigma^2_{ijk}}{n_{ijk}})$$

where μ_{ijk} is the sample mean, n_{ijk} is the sample size, and σ^2_{ijk} is the sample variance of observations for source port i, destination port j, and continuous feature k.

Substituting these values into the Gaussian probability density function yields the likelihood:

$$\frac{n_{ijk}}{\sigma^2_{ijk}} \sum (\bar{y}_{ijk} - \mu_{ijk})^2$$

Applying the CP/PARAFAC decomposition u_{ijk} is re-expressed:

$$u_{ijk} = \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$$

Vectorizing the inputs in the likelihood yields:

$$\sum_j \sum_k [\bar{y}_{ijk} - a_i^T (b_i \cdot c_k)] \frac{n_{ijk}}{\sigma^2_{ijk}} (1)$$

where $a_i \in \mathbb{R}_{\diagdown\times\diagdown}$, $b_j \in \mathbb{R}_{\diagdown\times\diagdown}$, and $c_k \in \mathbb{R}_{\diagdown\times\diagdown}$. Summing across j and k in this case solves for the ith row slice of the tensor. How to notate vectorization of y?

Recall the Residual Sum of Squares (RSS) of the likelihood for an Ordinary Least Squares (OLS) regression is expressed:

$$\sum_{l}(y_l - B^T x_l)^2$$

Adding a weight, w_l to the summation yields a Weighted Least Squares problem (WLS)

$$\sum_{l}^{n} w_l(y_l - \beta^T x_l)^2 (2)$$

that is analagous to the vectorized likelihood equation (1) with $w_l = \frac{n_{ijk}}{\sigma^2_{ijk}}$, $\beta = a_i$, $x = (b_j \cdot c_k)$.

With this formulation its now possible to solve for the optimal values for each slice a_i of the tensor.

Recall that in a traditional vectorized OLS, $y = X\beta + \sigma\epsilon$, where $y \in \mathbb{R}_{\aleph \times \aleph\aleph}$, $X \in \mathbb{R}_{\aleph \times \beth}$, $\beta \in \mathbb{R}_{\beth \times \aleph\aleph}$, and
$sigma\epsilon \in \mathbb{R}_{\aleph \times \aleph\aleph}$. Solving the maximum likelihood estimator of β, gives $\hat{\beta} = (X^T X)^{-1} X^T y$.

Applying this formulation to the weighted least squares gives $y = X\beta + W^{-\frac{1}{2}}\epsilon$. Solving for the weighted least squares estimator gives $\hat{\beta} = (X^T W X)^{-1} X^T W y$.

Repeating this estimation technique across each slice of the tensor a_i, b_j, c_k results in a completed model for y_{ijk}.

Chapter 4

Imputing Port Connections

4.1 Motivation

Following principal component analysis on the present ports combinations, the analysis shifts focus to the port combinations that are not present in the dataset. Imputing values for each of the four continuous features in the dataset for all possible source and destination port combinations yields a reasonable expected value in each cell of the ports matrix that can then be compared to actual connection values when they are observed. Actual values that differ greatly from the imputed values are flagged as anomalies and require further investigation.

This results in a 3-dimensional matrix completion problem. The matrix dimensions are defined as the number of source ports by the number of destination ports by the number of continuous features observed in each transaction. Similar techniques were employed in the Netflix Challenge where top competitors used matrix completion to predict ratings for movies by users that had not watched the movie based on the other entries in the matrix of users and movies.

4.2 Matrix Completion Algorithm

There are m source ports and n destination ports. $Y \in \mathbb{R}^{m \times n}$, is the matrix that stores the means of the combinations of source ports and destination ports. Y has missingness because not every source port interacts with every destination port. $F \in \mathbb{R}^{m \times n}$ is a sparse matrix that represents the frequencies of combinations, i.e $F[32242, 12312]$ represents the number of observations for the 32242 12312 port interaction. $M \in \mathbb{R}^{m \times n}$ represents a boolean matrix of whether the corresponding Y values are missing. $Y[M]$ represents all of the missing values of Y.

The objective is

$$min \sum_{i,j:F_{i,j}>0} (Y_{i,j} - u_i D v_j^T)^2$$

where UDV^T represents the singular value decomposition of Y. There are multiple steps to the matrix completion process:

4.2.1 Anova Initial Imputation

Impute the initial values for the missing $y_{i,j}$ observations $1 \leq i \leq m, 1 \leq j \leq n$: In general an additive model is applicable:

$$y_{i,j} = \mu + a_i + b_j + \epsilon_{i,j}$$

where $\epsilon \in N(0, \sigma^2)$, μ is the overall mean, a_i is the row mean, and b_j is column mean. An analysis of variance (ANOVA) imputation is used to fill in the initial values, $y_{i,j}$. Ignoring the missing values for now, let $y_{..}$ denote the empirical overall mean, $y_{i.}$ denote the empirical row mean, and $y_{.j}$ denote the column mean.

$$y_{i,j} = y_{..} + (y_{i.} - y_{..}) + (y_{.j} - y_{..}) = y_{i.} + y_{.j} - y_{..}$$

4.2.2 Repeated Imputation

The repeated imputation procedure solves $Y^{(s)}[M] = R_k(Y^{(s-1)})[M]$ where R_k is the best rank-k approximation for the s-th step. For each step (s) use singular value decomposition to decompose

$$Y^{(s)} = U^{(s)} D V^{T(s)}$$

where D is a diagonal matrix of the singular values, U is the left singular vectors of Y and V is the right singular vectors of Y.

The Eckart-Young-Mirsky (EYM) Theorem provides the best rank-k approximation for the missing values in $Y^{(s+1)}$. Recall $Y[M]$ represents all of the missing values of Y. Applying the EYM theorem:

$$Y^{(s+1)}[M] = (U[, 1:k]^{(s)} D[, 1:k] V[, 1:k]^{T(s)})[M]$$

Where $U[, 1:k]$ represents the first k columns of U and the same for D and V.

4.2.3 Convergence Criterion

The EYM rank approximation imputation steps are repeated until the relative difference between $Y^{(s+1)}$ and $Y^{(s)}$ falls below a set threshold, T. The relative difference threshold is expressed:

$$\frac{\|Y^{(s+1)} - Y^{(s)}\|_2}{\|Y^{(s)}\|_2} < T$$

where $\|Y\|_2$ is the Frobenius norm. The denominator of the expression ensures the convergence criterion is invariate to a scale change in the matrix itself.

4.2.4 Implementation

```
#matrix parameters
n_Sport = 20
n_Dport = 20

#get freqs
Sport_table = as.data.frame(table(argus$Sport))
Sport_table = Sport_table[order(-Sport_table$Freq),]
top_Sport = (head(Sport_table$Var1, n_Sport))

#get freqs
Dport_table = as.data.frame(table(argus$Dport))
Dport_table = Dport_table[order(-Dport_table$Freq),]
top_Dport = (head(Dport_table$Var1, n_Dport))

#create starting matrices
ports_combo_matrix = matrix(list(), nrow = n_Sport, ncol = n_Dport)
dimnames(ports_combo_matrix) = list(top_Sport, top_Dport)

ports_freq_matrix = matrix(0, nrow = n_Sport, ncol = n_Dport)
dimnames(ports_freq_matrix) = list(top_Sport, top_Dport)

#fill the ports_combo_matrix and ports_freq_matrix
for (s in 1:n_Sport){
  for (d in 1:n_Dport){
    combination = argus[is.element(argus$Sport, top_Sport[s])
                        & is.element(argus$Dport, top_Dport[d]),]
    obs = combination$SrcBytes
    n_obs = length(obs) #ignores NA values
    if (n_obs > 0){
      obs = nscore(obs)$nscore #normal transformation
      for (i in 1:n_obs){
        ports_combo_matrix[[s,d]] = c(ports_combo_matrix[[s,d]],obs[i])
        #O(1) time to append values to a list?
        ports_freq_matrix[s,d] = ports_freq_matrix[s,d] + 1
      }
    }
  }
}

#create mean and variance matrix
ports_mean_matrix = matrix(NA, nrow = n_Sport, ncol = n_Dport)
dimnames(ports_mean_matrix) = list(top_Sport, top_Dport)

ports_variance_matrix = matrix(NA, nrow = n_Sport, ncol = n_Dport)
```

```
dimnames(ports_variance_matrix) = list(top_Sport, top_Dport)

#fill mean and variance matrix
for (s in 1:n_Sport){
  for (d in 1:n_Dport){
    if (ports_freq_matrix[s,d] == 1){
      ports_mean_matrix[s,d] = ports_combo_matrix[[s,d]]
      ports_variance_matrix[s,d] = 0
    }
    else if (ports_freq_matrix[s,d] > 1){
      ports_mean_matrix[s,d] = mean(ports_combo_matrix[[s,d]])
      ports_variance_matrix[s,d] = var(ports_combo_matrix[[s,d]])
    }
  }
}

####Eckhart Young Theorem Implementation, Best Rank k Approximation####
matrix_complete = function(S = 1000, k = 2, n_Sport, n_Dport, Y, M){
  S = 1000
  k = 2
  Y_imputed = Y
  #calculate overall mean
  n = 0
  sum = 0
  for (s in 1:n_Sport){
    for (d in 1:n_Dport){
      if (M[s,d] != 0){
        sum = sum + Y[s,d]
        n = n + M[s,d]
      }
    }
  }
  overall_mean = sum/n
  #calculate row means and col means
  row_means = rowMeans(Y, na.rm = TRUE)
  col_means = colMeans(Y, na.rm = TRUE)
  #set NaN to 0 in means to fix anova fill in
  for (i in 1:n_Sport){
    if (!is.finite(row_means[i])){
      row_means[i] = 0
    }
    if (!is.finite(col_means[i])){
      col_means[i] = 0
    }
```

```
  }
  #Fill in missing values in Y_imputed with ANOVA
  for (s in 1:n_Sport){
    for (d in 1:n_Dport){
      if (M[s,d] == 0){
        Y_imputed[s,d] = row_means[s] + col_means[d] - overall_mean
      }
    }
  }
  for (i in 1:S){
    #extract SVD
    svd_Y = svd(Y_imputed)
    D = diag((svd_Y$d)[1:k])
    U = svd_Y$u
    V = svd_Y$v
    #EYM theorem
    EYM = U[,1:k] %*% D %*% t(V[,1:k])
    for (s in 1:n_Sport){
      for (d in 1:n_Dport){
        if (M[s,d] == 0){
          Y_imputed[s,d] = EYM[s,d]
        }
      }
    }
  }
  return (Y_imputed)
}

ports_mean_matrix_imputed = matrix_complete(1000, 2, n_Sport, n_Dport, ports_mean_

#Relative distance using Frobenius Norm
relative_distance = function(Y, Y_imputed){
  return (sqrt(sum((Y - Y_imputed)^2)) / sqrt(sum(Y^2)))
}
```

4.3 Assessing Imputation Strategy

4.3.1 Leave One Out Cross Validation

To assess the quality of the imputation, Leave-One-Out Cross Validation (LOOCV) is used to generate a prediction error. LOOCV cycles through the observed values, setting each to NA (missing), and then performing the described imputation process. The prediction error is then calculated as some function of the difference between the imputed value and the true value. In this case, the algorithm records absolute error

$\sum \mid \hat{y}_{i,j} - y_{i,j} \mid$ and root mean square error $\sqrt{\frac{\sum(\hat{y}_{i,j}-y_{i,j})^2}{n}}$ where n is the number of observations not missing.

4.3.2 Implementation

```
#Leave One Out Cross Validation
loocv = function (S = 1000, k = 2, nrows = n_Sport, ncols = n_Dport, Y, M){
  error = 0
  rmse = 0
  n = 0
  for (s in 1:nrows){
    for (d in 1:ncols){
      if (M[s,d] != 0){
        n = n + 1
        M_imputed = M
        M_imputed[s,d] = 0
        Y_imputed = matrix_complete(S, k, nrows, ncols, Y, M_imputed)
        error = error + abs((Y_imputed[s,d] - Y[s,d]))
        rmse = rmse + (Y_imputed[s,d] - Y[s,d])^2
      }
    }
  }
  rmse = sqrt(rmse/n)
  return (list(Error = error, RMSE = rmse, Observations = n))
}
#find optimal rank
```

4.3.3 Results

While matrix completion via singular value decomposition presents valid missing value imputations, and the algorithm converges relatively quickly, the error generated from leave one out cross validation reflects that the imputation performs rather poorly for low rank solutions to the data. Moreover, the errors are minimized at a rank approximation of 3, but even at this rank, the errors are relatively high considering the data was first normal transformed.

This poor performance may largely be due to the fact the algorithm does not account for the variability in the number of observed solutions for each cell being imputed. Unlike the Netflix Competition, in which each cell of the matrix being completed contained only a single user rating of a movie, the matrix in this problem contains the average of a variable number of observations in each cell.

Chapter 5

Statistical Model for Port Behavior

5.1 Motivation

The previous section's results reflected the need for an imputation strategy that accounted for the variability in the number of observations observed for each port combination when imputing that particular combination's cell. The following section constructs a statistical model that takes frequency of observations for each cell into account and repeatedly samples from that statistical model to complete the matrix.

5.2 AMMI Model

Additive Main Effects and Multiplicative Interaction Models (AMMI models) provide a defined statistical model for each cell in the ports matrix. In particular, the model combines the additive effects of the initial ANOVA imputation with the multiplicative effects yielded from singular value decomposition described in the previous section. More importantly, the model also includes a variance term for each cell that takes into account the differing frequency of observations in each port combination. Applying the same mathematical notation as the previous section, the model is formally expressed:

$$y_{i,j} = u + a_i + b_j + \mathbf{u_i} D \mathbf{v_j^T} + \sigma_{i,j} \epsilon_{i,j}$$

where $sigma_{i,j}$ is the variance for the ith row jth column in the ports combination matrix, and $\epsilon \sim N(0, 1)$.

5.3 Gibbs Sampling

Following the application of the AMMI model, Gibbs Sampling is used to repeatedly generate samples from the statistical model, yielding an approximate value for each cell.

Conclusion

If we don't want Conclusion to have a chapter number next to it, we can add the `{-}` attribute.

More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

Appendix A

Preliminary Data Investigation

A.1 Exploratory Data Analysis

A.1.1 Cleaning Predictors

```
sapply(argus, class)
```

```
    Flgs   SrcAddr     Sport   DstAddr     Dport   SrcPkts   DstPkts
"factor"  "factor"  "factor"  "factor"  "factor" "integer" "integer"
 SrcBytes  DstBytes     State
"integer" "integer"  "factor"
```

```
argus = transform(argus,
                  Sport = as.factor(argus$Sport),
                  Dport = as.factor(argus$Dport))
argus = subset(argus, select = c("Flgs", "SrcAddr", "Sport", "DstAddr", "Dport",
                                 "SrcPkts", "DstPkts", "SrcBytes", "DstBytes", "Sta
attach(argus)
```

```
The following objects are masked from argus (pos = 3):

    Dport, DstAddr, DstBytes, DstPkts, Flgs, Sport, SrcAddr,
    SrcBytes, SrcPkts, State
```

```
categorical = c("Flgs", "SrcAddr", "Sport", "DstAddr", "Dport", "State")
continuous = c("SrcPkts", "DstPkts", "SrcBytes", "DstBytes")
```

This code casts the features to their corresponding class classifications (numeric and factor), and removes Proto, StartTime, and Diretion from the dataset.

A.1.2 Categorical Features: Unique Categories and Counts

```
sapply(argus, function(x) length(unique(x)))
```

```
   Flgs  SrcAddr    Sport  DstAddr    Dport  SrcPkts  DstPkts SrcBytes
     70    65113    64486    41903    17537     2790     3633    44075
DstBytes    State
   64549        6
```

```
#function that returns elements of the feature and their counts in descending order
element_counts = function(x) {
  dt = data.table(x)[, .N, keyby = x]
  dt[order(dt$N, decreasing = TRUE),]
}
element_counts(Sport)
```

```
          x     N
   1: 33461 35683
   2:  4263  8541
   3:    80  8346
   4:  4165  4988
   5: 48468  3023
  ---
64482: 57904     1
64483: 58242     1
64484: 59051     1
64485: 59315     1
64486: 60116     1
```

```
element_counts(Dport)
```

```
          x      N
   1:    23 235616
   2:    80 204128
   3:   443 137360
   4: 32819 102166
   5:    25  53167
  ---
17533: 65515      1
17534: 65528      1
17535: 65529      1
17536: 65532      1
17537: 65533      1
```

```
element_counts(SrcAddr)
```

```
                x     N
   1: 197.0.31.231 35440
   2:    1.0.11.96  8717
   3:  100.0.7.149  8526
   4:    197.0.9.1  5536
   5:   1.0.85.103  4976
  ---
65109:   197.0.55.5     1
65110:   197.0.55.6     1
65111:   197.0.55.7     1
65112:   197.0.55.8     1
65113:   197.0.55.9     1
```

```
element_counts(DstAddr)
```

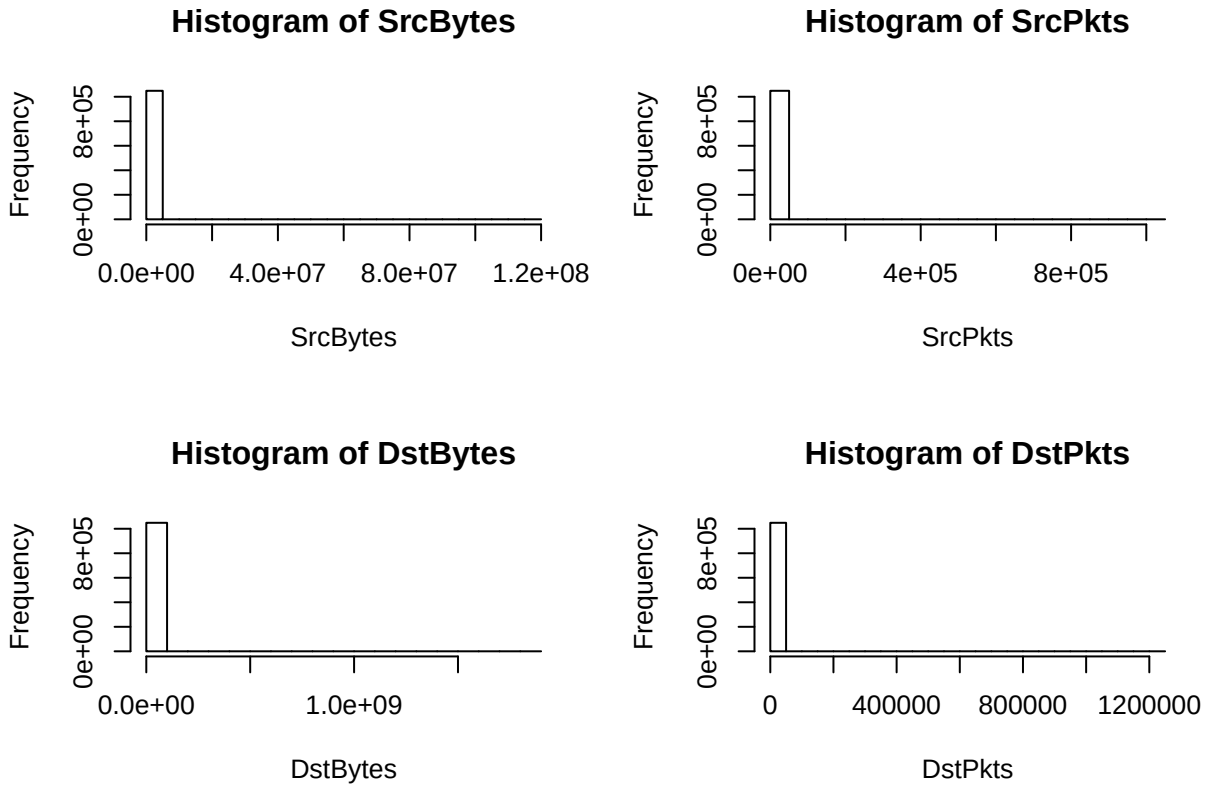
```
                x     N
   1:    100.0.1.9 64508
   2:    100.0.1.2 62681
   3:   100.0.1.28 25780
   4:   100.0.1.55 25641
   5:  100.0.18.93 20766
  ---
41899:  100.0.6.179     1
41900:  100.0.7.173     1
41901: 100.0.77.113     1
41902: 100.0.77.147     1
41903: 100.0.88.111     1
```

```
element_counts(State)
```

```
    x      N
1: REQ 571899
2: FIN 307337
3: RST 135280
4: CON  17146
5: ACC  16815
6: CLO     98
```

A.1.3 Continuous Features: Distributions and Relationships

```
par(mfrow=c(2,2))
hist(SrcBytes); hist(SrcPkts); hist(DstBytes); hist(DstPkts) #clearly some very large v
```




```
largest_n = function(x, n){
  head(sort(x, decreasing=TRUE), n)
}
largest_n(SrcBytes, 10)
```

```
[1] 118257047 116615879 112673526 108933442 105793666  73376579  72839115
[8]  70001807  56206409  55359912
```

```
largest_n(SrcPkts, 10)
```

```
[1] 1008233 1000971  771590  492361  458603  437296  408530  407973
[9]  371976  371251
```

```
largest_n(DstBytes, 10)
```

```
[1] 1850817751 1713055847 1690162763 1524781880 1491609296 1340922625
[7] 1304668214 1206594243 1163954979 1145323438
```

```
largest_n(DstPkts, 10)
```

```
[1] 1239611 1223485 1219276 1004471  982931  942827  883354  795120
[9]  766776  754831
```

The histograms and the largest 10 values in each of the continuous variables show that there are a relatively few amount of large observations skewing the distributions. This explains the model summary containing means much larger than their medians. It's not possible to remove the large values as outliers because they may be scanner observations to detect. Also there is a high frequency (up to the first quartile) of destination bytes and packets that equal 0.

We will now try to investigate whether the largest continuous predictor values correspond to any particular addresses or ports.

```
max.SrcBytes = argus[with(argus,order(-SrcBytes)),][1:20,]
max.SrcPkts = argus[with(argus,order(-SrcPkts)),][1:20,]
max.DstBytes = argus[with(argus,order(-DstBytes)),][1:20,]
max.DstPkts = argus[with(argus,order(-DstPkts)),][1:20,]
head(max.SrcBytes)
```

```
            Flgs   SrcAddr Sport    DstAddr Dport SrcPkts DstPkts
282859  * s        1.0.12.1 18086  100.0.1.8 31743  208339  104886
282841  * s        1.0.12.1 18086  100.0.1.8 31743  204912   99688
282832  * s        1.0.12.1 18086  100.0.1.8 31743  198007   94621
282853  * s        1.0.12.1 18086  100.0.1.8 31743  191443   95892
282823  * s        1.0.12.1 18086  100.0.1.8 31743  186228   84342
724162  * *       100.0.4.9 37901 100.0.2.67    22 1008233 1239611
       SrcBytes   DstBytes State
282859 118257047    7514403   CON
282841 116615879    7146182   CON
282832 112673526    6801146   CON
282853 108933442    6857053   CON
282823 105793666    6048529   CON
724162  73376579 1713055847   CON
```

```
head(max.DstBytes)
```

```
            Flgs   SrcAddr Sport    DstAddr Dport SrcPkts DstPkts
2162    * d       197.0.1.1 62030  100.0.1.1    80  371251 1219276
724162  * *       100.0.4.9 37901 100.0.2.67    22 1008233 1239611
724106  * *       100.0.4.9 37901 100.0.2.67    22 1000971 1223485
2212    * d       197.0.1.1 62034  100.0.1.1    80  280593 1004471
78245   * d         1.0.2.1 11210  100.0.1.2    80  158194  982931
2185    * d       197.0.1.1 62033  100.0.1.1    80  268402  883354
```

```
       SrcBytes   DstBytes State
2162   26430322 1850817751   CON
724162 73376579 1713055847   CON
724106 72839115 1690162763   CON
2212   20037592 1524781880   FIN
78245  11294111 1491609296   CON
2185   19137354 1340922625   FIN
```

Source Addresses tend to be repetitive for the largest max bytes/packets, while ports vary. The top 10 largest DstBytes all correspond to SrcAddr 197.0.1.1 and DstAddr 100.0.1.1. Also both max Src and Dst rows correspond to the "* s"" flag. The largest sizes of DstBytes tend to go to Dport 80, which is the port that expects to receive from a web client (http), while the largest SrcBytes go to 31743. The next section implements a systematic way for investigating the relationship between addresses and ports because simply looking at the max rows is difficult.

A.1.4 Correlation Between Features

```
cor(SrcBytes, SrcPkts)
```

```
[1] 0.5732968
```

```
cor(DstBytes, DstPkts)
```

```
[1] 0.996775
```

```
cor(SrcBytes, DstBytes)
```

```
[1] 0.3331221
```

```
cor(SrcPkts, DstPkts)
```

```
[1] 0.8448269
```

The plots of the predictors suggest strong linear trends between the predictors, which makes intuitive sense given the domain matter. Further investigations show that DstPkts has a correlation of ~1 with DstBytes and ~0.85 with SrcPkts.

```
cor(DstBytes, DstPkts, method = "kendall")
cor(SrcPkts, DstPkts, method = "kendall")
cor(DstBytes, DstPkts, method = "spearman")
cor(SrcPkts, DstPkts, method = "spearman")
```

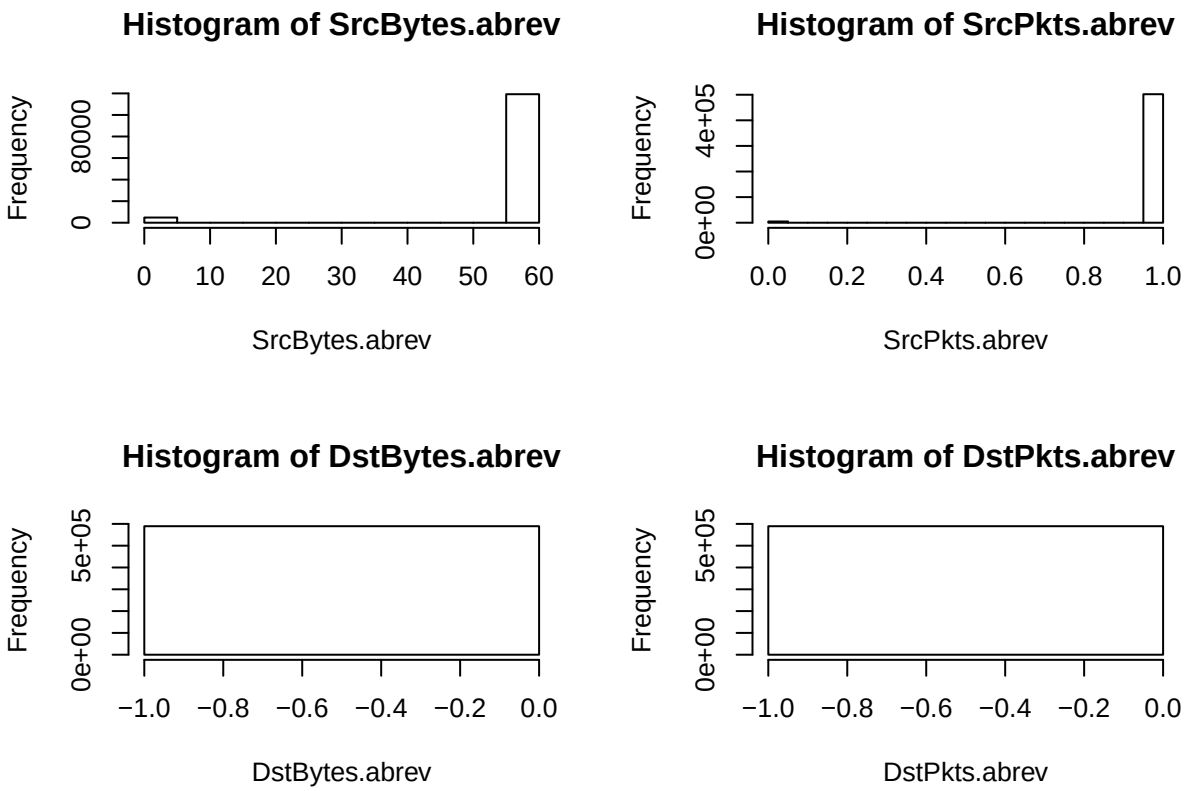
Because the original correlation tests relied on the Pearson method, which is susceptible to bias from large values, further tests investigate the relationship between DstPkts and the other features. While the correlation is still high for Kendall-Tau and Spearman's correlation coefficients, it is less cause for concern when compared to the skewed response from the Pearson method.

A.2 Transformations on the Data

A.2.1 Removing Quantiles

To get a better sense of the unskewed distribution, the below plots visualize the continuous features with the largest and smallest 10% of observations removed. The removed values will be readded to the dataset when investigating for anomalies.

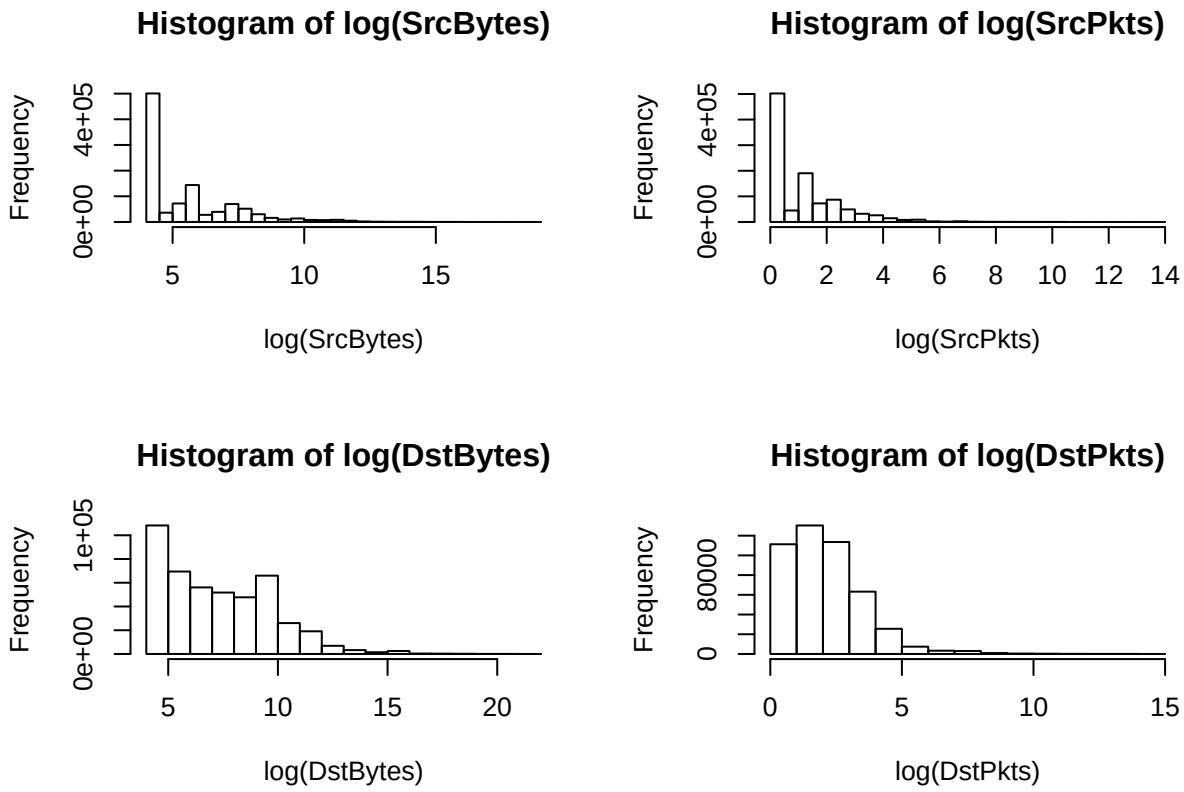
```
remove_quantiles = function(v, lowerbound, upperbound){
  return (v[quantile(v,lowerbound) >= v & v <= quantile(v,upperbound)])
}
SrcBytes.abrev = remove_quantiles(SrcBytes,0.10,0.9)
SrcPkts.abrev = remove_quantiles(SrcPkts,0.10,0.9)
DstBytes.abrev = remove_quantiles(DstBytes,0.10,0.9)
DstPkts.abrev = remove_quantiles(DstPkts,0.10,0.9)
par(mfrow=c(2,2))
hist(SrcBytes.abrev); hist(SrcPkts.abrev); hist(DstBytes.abrev); hist(DstPkts.abre
```



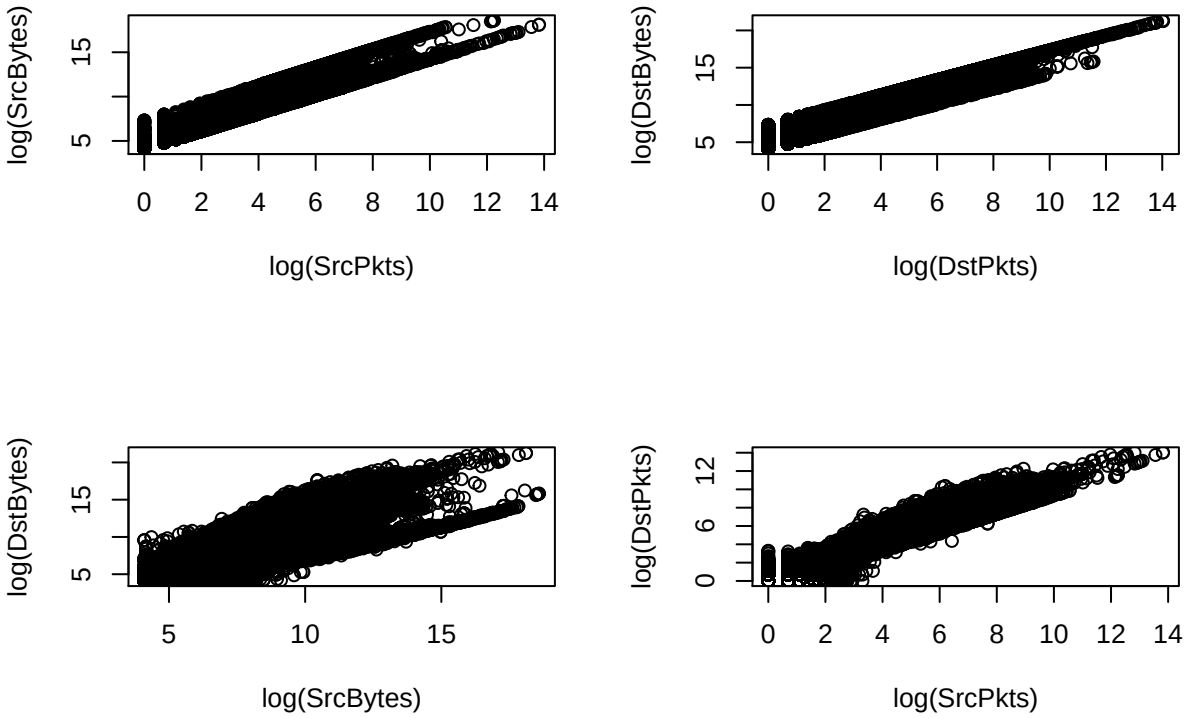
The continuous features are still unevenly distributed even with the 20% most extreme values removed.

A.2.2 Log Transformation

```
par(mfrow=c(2,2))
hist(log(SrcBytes)); hist(log(SrcPkts)); hist(log(DstBytes)); hist(log(DstPkts))
```




```
plot(log(SrcPkts), log(SrcBytes)); plot(log(DstPkts), log(DstBytes))
plot(log(SrcBytes), log(DstBytes)); plot(log(SrcPkts), log(DstPkts))
```



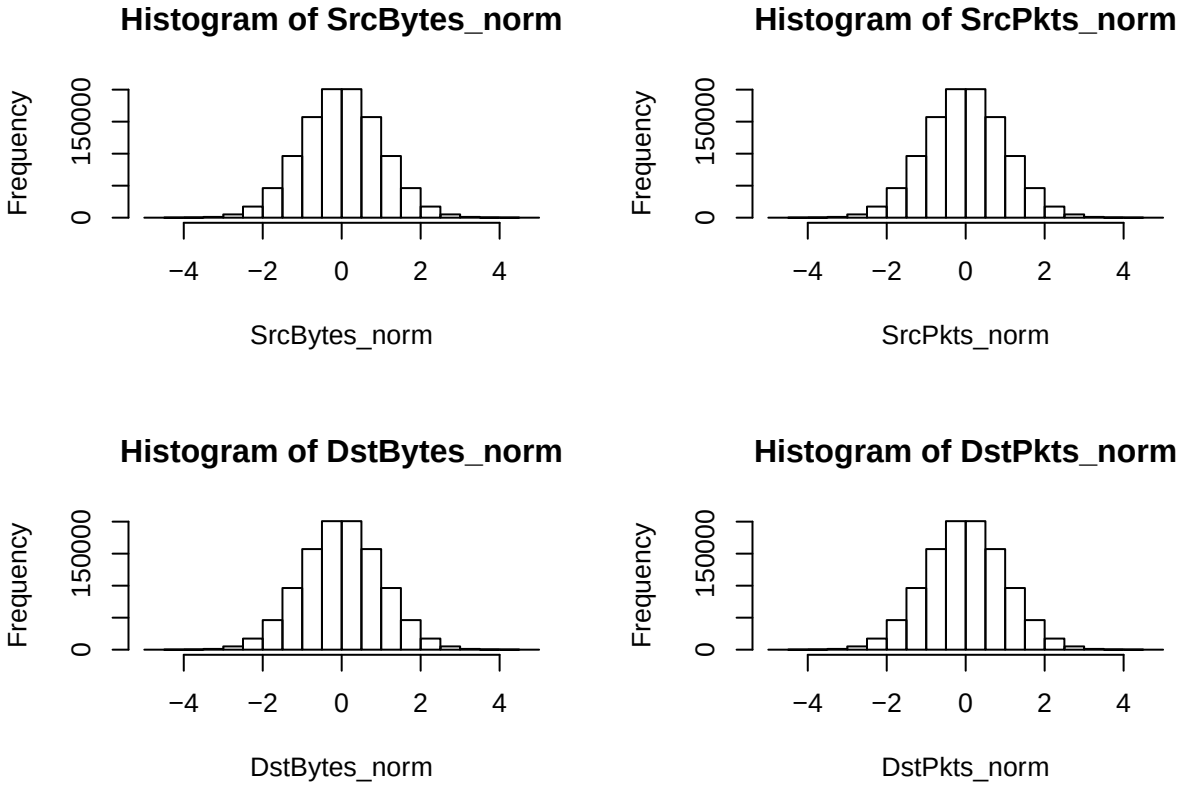
A log transformation for each of the continuous features outputs right-skewed histograms. Skewed features may affect the results of a kernel pca, so we consider other approaches for transformations.

A.2.3 Normal Scores Transformation

```
nscore = function(x) {
   # Takes a vector of values x and calculates their normal scores. Returns
   # a list with the scores and an ordered table of original values and
   # scores, which is useful as a back-transform table. See backtr().
   nscore = qqnorm(x, plot.it = FALSE)$x  # normal score
   trn.table = data.frame(x=sort(x),nscore=sort(nscore))
   return (list(nscore=nscore, trn.table=trn.table))
}

backtr = function(scores, nscore, tails='none', draw=TRUE) {
   # Given a vector of normal scores and a normal score object
   # (from nscore), the function returns a vector of back-transformed
   # values
   # 'none' : No extrapolation; more extreme score values will revert
   # to the original min and max values.
   # 'equal' : Calculate magnitude in std deviations of the scores about
   # initial data mean. Extrapolation is linear to these deviations.
   # will be based upon deviations from the mean of the original
   # hard data - possibly quite dangerous!
```

```
   # 'separate' :  This calculates a separate sd for values
   # above and below the mean.
   if(tails=='separate') {
      mean.x <- mean(nscore$trn.table$x)
      small.x <- nscore$trn.table$x < mean.x
      large.x <- nscore$trn.table$x > mean.x
      small.sd <- sqrt(sum((nscore$trn.table$x[small.x]-mean.x)^2)/
                       (length(nscore$trn.table$x[small.x])-1))
      large.sd <- sqrt(sum((nscore$trn.table$x[large.x]-mean.x)^2)/
                       (length(nscore$trn.table$x[large.x])-1))
      min.x <- mean(nscore$trn.table$x) + (min(scores) * small.sd)
      max.x <- mean(nscore$trn.table$x) + (max(scores) * large.sd)
      # check to see if these values are LESS extreme than the
      # initial data - if so, use the initial data.
      #print(paste('lg.sd is:',large.sd,'max.x is:',max.x,'max nsc.x
      #     is:',max(nscore$trn.table$x)))
      if(min.x > min(nscore$trn.table$x)) {min.x <- min(nscore$trn.table$x)}
      if(max.x < max(nscore$trn.table$x)) {max.x <- max(nscore$trn.table$x)}
   }
   if(tails=='equal') { # assumes symmetric distribution around the mean
      mean.x <- mean(nscore$trn.table$x)
      sd.x <- sd(nscore$trn.table$x)
      min.x <- mean(nscore$trn.table$x) + (min(scores) * sd.x)
      max.x <- mean(nscore$trn.table$x) + (max(scores) * sd.x)
      # check to see if these values are LESS extreme than the
      # initial data - if so, use the initial data.
      if(min.x > min(nscore$trn.table$x)) {min.x <- min(nscore$trn.table$x)}
      if(max.x < max(nscore$trn.table$x)) {max.x <- max(nscore$trn.table$x)}
   }
   if(tails=='none') {   # No extrapolation
      min.x <- min(nscore$trn.table$x)
      max.x <- max(nscore$trn.table$x)
   }
   min.sc <- min(scores)
   max.sc <- max(scores)
   x <- c(min.x, nscore$trn.table$x, max.x)
   nsc <- c(min.sc, nscore$trn.table$nscore, max.sc)

   if(draw) {plot(nsc,x, main='Transform Function')}
   back.xf <- approxfun(nsc,x) # Develop the back transform function
   val <- back.xf(scores)
   return(val)
}
```

```
SrcBytes_norm = nscore(SrcBytes)$nscore
SrcBytes_table = nscore(SrcBytes)$trn.table

SrcPkts_norm = nscore(SrcPkts)$nscore
SrcPkts_table = nscore(SrcPkts)$trn.table

DstBytes_norm = nscore(DstBytes)$nscore
DstBytes_table = nscore(DstBytes)$trn.table

DstPkts_norm = nscore(DstPkts)$nscore
DstPkts_table = nscore(DstPkts)$trn.table
par(mfrow=c(2,2))
hist(SrcBytes_norm); hist(SrcPkts_norm); hist(DstBytes_norm); hist(DstPkts_norm)
```



Finally, a normal scores transformation is applied to the dataset. The normal scores transformation reassigns each feature value so that it appears the overall data for that feature had arisen or been observed from a standard normal distribution. This transformation solves the issue of skewness-each value's histogram will now follow a standard gaussian density plot-, but it may cause issues with other analysis methods, particularly methods that are susceptible to ties in data.

References

Angel, Edward. 2000. *Interactive Computer Graphics : A Top-down Approach with Opengl*. Boston, MA: Addison Wesley Longman.

———. 2001a. *Batch-File Computer Graphics : A Bottom-up Approach with Quicktime*. Boston, MA: Wesley Addison Longman.

———. 2001b. *Test Second Book by Angel*. Boston, MA: Wesley Addison Longman.