Tensor Completion Techniques For Anomaly Detection in Network Attacks

---

A Thesis
Presented to
Department of Statistical Science
Duke University

James C. Wu

May 2018

Approved for the
Bachelor of Science in Statistical Science

_____

Peter D. Hoff

_____

Committeemember O. Name

_____

Committeemember T. Name

_____

Mine Cetinkaya-Rundel, DUS

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.

Table of Contents

Abstract

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.

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

Preliminary 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.

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.

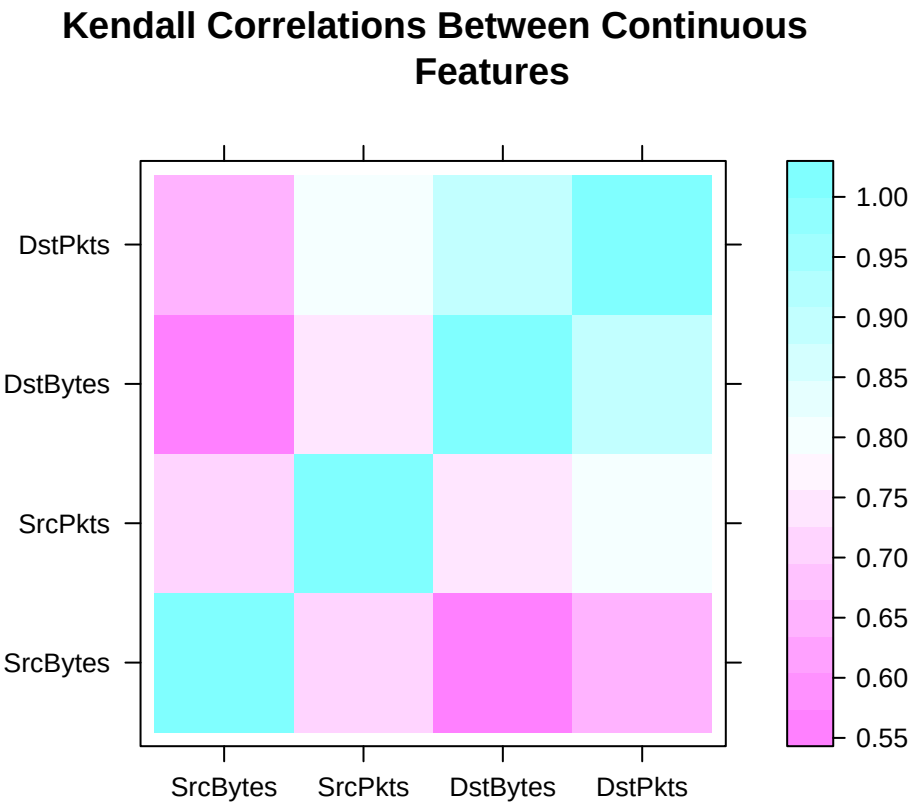
The goal of this project is to devise an optimal strategy for imputing the missing cells in Y to create the completed tensor $Y' \in \mathbb{R}_{\geqslant \times \leqslant \times \not\geqslant}$. As new observations are observed for combinations of ports i and j, the y'_{ijk} values can be interpreted 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.

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. New observations that differ greatly from the imputed values are flagged as anomalies and require further investigation.

2.2 Tensor Properties

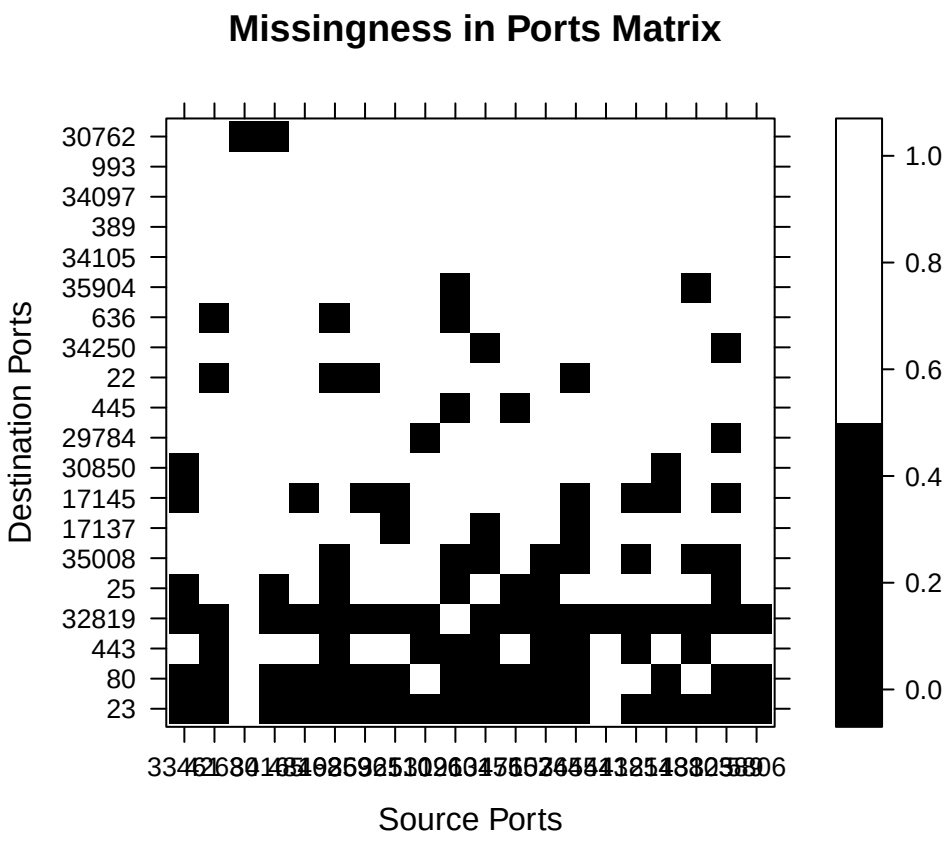
The following properties of Y inform the imputation strategies in following sections.

2.2.1 Correlations

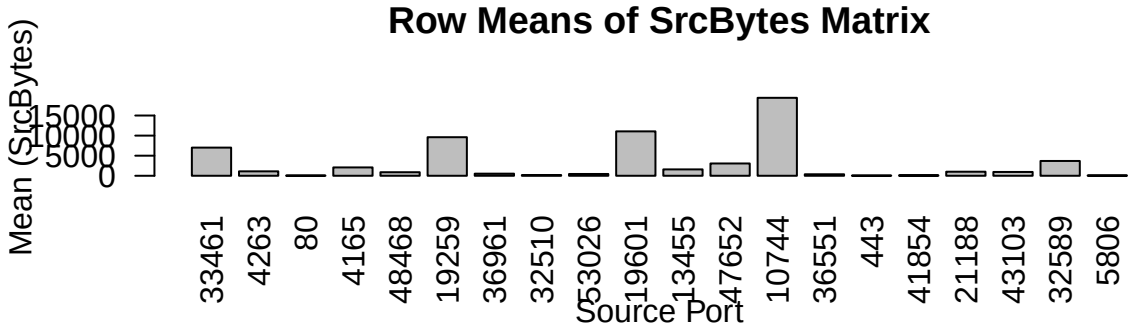


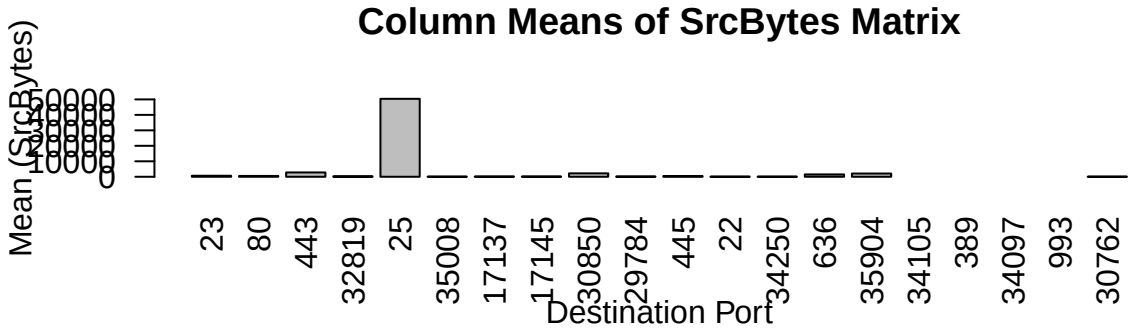
2.2.2 Missingness

```
[1] 295
```


Missingness in Ports Matrix
Destination Ports
30762
993
34097
389
34105
35904
636
34250
22
445
29784
30850
17145
17137
35008
25
32819
443
80
23
Source Ports
1.0
0.8
0.6
0.4
0.2
0.0

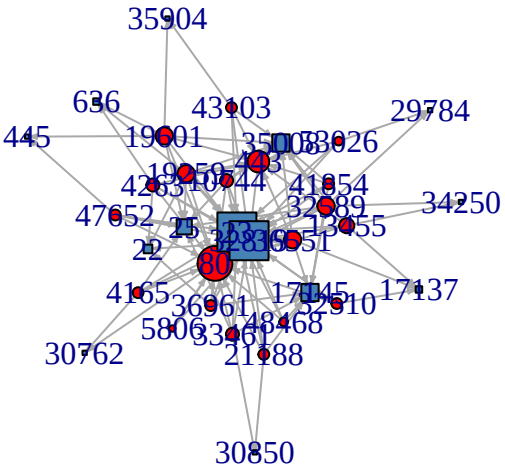
2.2.3 Row and Column Properties





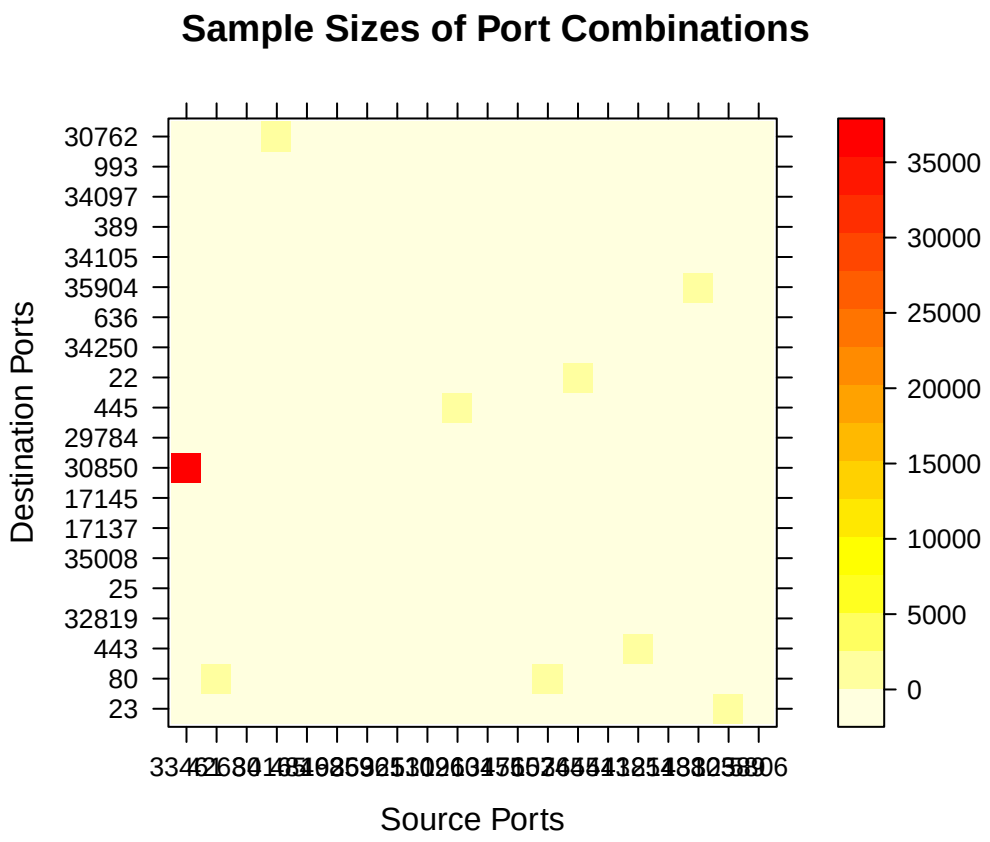
2.2.4 Port Connections

```
Attaching package: 'igraph'

The following objects are masked from 'package:dplyr':

    as_data_frame, groups, union

The following objects are masked from 'package:stats':

    decompose, spectrum

The following object is masked from 'package:base':

    union
```



2.2.5 Matrix Slice Properties

```
### MATRIX VISUALIZATIONS
# https://stackoverflow.com/questions/5453336/plot-correlation-matrix-into-a-gra
# levelplot(means, xlab = "Source Ports", ylab = "Destination Ports",
#           main = "Means of SrcBytes By Ports",
#           col.regions=rgb.palette(120))
levelplot(freqs, xlab = "Source Ports", ylab = "Destination Ports",
          col.regions = heat.colors(16)[length(heat.colors(16)):1],
          main = "Sample Sizes of Port Combinations")
```


Sample Sizes of Port Combinations
Destination Ports
30762
993
34097
389
34105
35904
636
34250
22
445
29784
30850
17145
17137
35008
25
32819
443
80
23
Source Ports
35000
30000
25000
20000
15000
10000
5000
0

Chapter 3

Alternating Least Squares Applied to Matrix Slices

3.1 Motivation

The preliminary tensor imputation technique slices the tensor into four separate matrices, $Y^{(1)}, Y^{(2)}, Y^{(3)}, Y^{(4)} \in \mathbb{R}_{m \times n}$. Each matrix has dimensions defined by the number of source ports by the number of destination ports. Each cell, $Y_{ij}^{(k)}$ contains the mean of the observations observed for continuous feature k and source port i destination port j. Each matrix will have its missing cells imputed with information from the rest of that matrix, and separately from information about the other matrices. The four completed matrices will then be recombined to create the tensor Y. Similar techniques for matrix completion were employed in the Netflix Challenge where top competitor predicted ratings for movies by users that had not watched the movie based on the other ratings in the matrix of users and movies.

3.2 Matrix Completion Algorithm

Each $Y^{(k)}$ 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. $Y^{(k)}[M]$ represents all of the missing values of $Y^{(k)}$. Moreover because each non-missing observation contains all four continuous features, $Y[M]$ represents all the missing values of Y.

The objective is

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

where $U^{(k)}D^{(k)}V^{(k)T}$ represents the singular value decomposition of $Y^{(k)}$. There are multiple steps to the matrix completion process:

3.2.1 Anova Initial Imputation

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

$$y_{i,j}^{(k)} = \mu^{(k)} + a_i^{(k)} + b_j^{(k)} + \epsilon_{i,j}^{(k)}$$

where $\epsilon^{(k)} \in N(0, \sigma^2)$, $\mu^{(k)}$ is the overall mean of $Y^{(k)}$, $a_i^{(k)}$ is the row mean, and $b_j^{(k)}$ is column mean. An analysis of variance (ANOVA) imputation is used to fill in the initial values, $y_{i,j}^{(k)}$. 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.

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

3.2.2 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

$$Y^{(s)} = U^{(s)}DV^{T(s)}$$

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.

3.2.3 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.

3.2.4 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)))
}
```

3.3 Assessing Imputation Strategy

3.3.1 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.

3.3.2 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
```

```
setwd("~/Desktop/Stats Thesis/")
argus = readRDS("Dataset/argus_complete.rds")

#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)

nscore = function(x) {
  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))
}

#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]])
    }
  }
}

#untuned ALS using softimpute
library(softImpute)
```

```
Loading required package: Matrix


Attaching package: 'Matrix'

The following object is masked from 'package:reshape':

    expand

Loaded softImpute 1.4
```

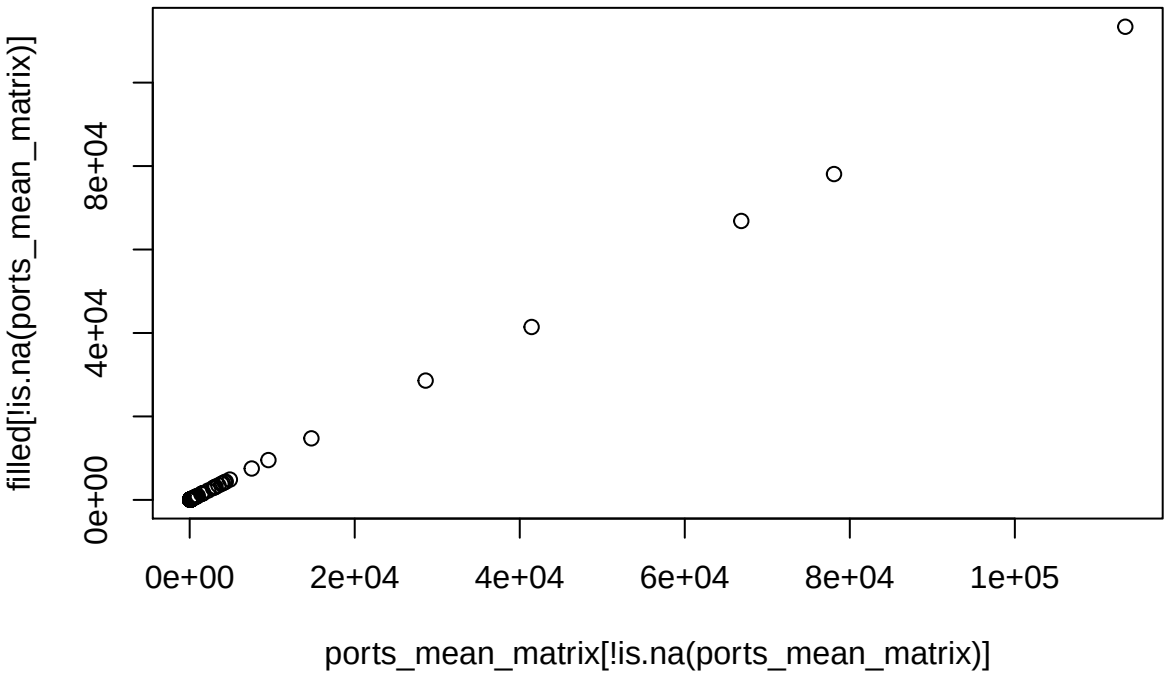
```
fit = softImpute(ports_mean_matrix,rank.max=3,lambda=0.9,trace=TRUE,type="als")
```

```
1 : obj 118447630 ratio 8862771621
2 : obj 338300.9 ratio 0.002398703
3 : obj 261856.6 ratio 0.0002617471
4 : obj 216131.7 ratio 0.0003460135
5 : obj 168096 ratio 0.0002697307
6 : obj 139246.7 ratio 0.0001180686
7 : obj 124787.1 ratio 6.149304e-05
8 : obj 115182.9 ratio 4.595611e-05
9 : obj 107326.8 ratio 3.952473e-05
10 : obj 100581.7 ratio 3.386247e-05
11 : obj 95056.82 ratio 2.698493e-05
12 : obj 90911.68 ratio 1.958786e-05
13 : obj 88054.62 ratio 1.316015e-05
14 : obj 86180.49 ratio 8.536606e-06
final SVD: obj 84890.87
```

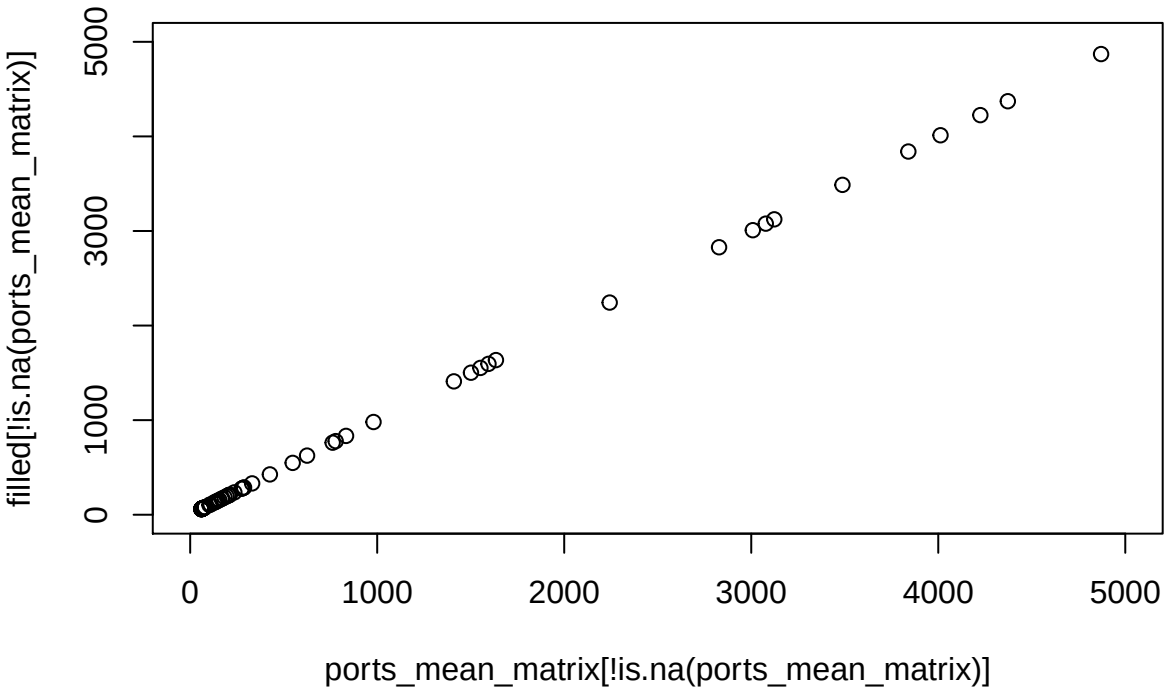
```
fit$d
```

```
[1] 163274.716  17272.466   7494.707
```

```
filled = complete(ports_mean_matrix, fit)
plot(ports_mean_matrix[!is.na(ports_mean_matrix)], filled[!is.na(ports_mean_matrix
```




```
plot(ports_mean_matrix[!is.na(ports_mean_matrix)], filled[!is.na(ports_mean_matrix
     xlim = c(0,5000), ylim = c(0,5000))
```




```
####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])
  #replace imputed values with new values, REPLACE ONLY MISSING OR REPLACE ALL?
```

```
    #Replacing only missing means we cant assess fitted error
    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)
}

matrix_complete2 = function(Y, k = 2, lambda = 1.0){
  fit = softImpute(Y, rank.max=k, lambda = lambda, trace=TRUE, type="als")
  # fit$d
  filled = complete(Y, fit)
  return(filled)
}

#ports_mean_matrix_imputed = matrix_complete(1000, 2, n_Sport, n_Dport, ports_me

#Relative distance using Frobenius Norm
relative_distance = function(Y, Y_imputed){
  return (frobenius.norm(Y - Y_imputed) / frobenius.norm(Y))
}

#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)
        true_sd = Y[s,d]
        Y[s,d] <- NA
        Y_imputed = matrix_complete2(Y, k, 0.9)
        error = error + abs((Y_imputed[s,d] - true_sd))
        rmse = rmse + (Y_imputed[s,d] - true_sd)^2
        Y[s,d] = true_sd
```

```
      }
    }
  }
  rmse = sqrt(rmse/n)
  return (list(Error = error, RMSE = rmse, Observations = n))
}

# ranks = seq(1,10,1)
# #RMSEs = lapply(seq(1,10,1), function(k) loocv(250,k,n_Sport, n_Dport, ports_mean_ma
# RMSEs = c()
# for (k in ranks){
#   print (k)
#   cv = loocv(250,k,n_Sport, n_Dport, ports_mean_matrix, ports_freq_matrix)
#   RMSEs = c(RMSEs, cv$RMSE)
# }
# plot(ranks, RMSEs)
#
# cv1 = loocv(250,1,n_Sport, n_Dport, ports_mean_matrix, ports_freq_matrix)
# cv2 = loocv(250,2,n_Sport, n_Dport, ports_mean_matrix, ports_freq_matrix)
# cv3 = loocv(250,3,n_Sport, n_Dport, ports_mean_matrix, ports_freq_matrix)
# cv5 = loocv(250,5,n_Sport, n_Dport, ports_mean_matrix, ports_freq_matrix)
```

3.3.3 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 4

Tensor Completion

4.1 Imputation Strategy

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

4.1.1 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 $u \in \mathbb{R}_{\gg \times \diagdown}$, $v \in \mathbb{R}_{\ltimes \times \diagdown}$, and $w \in \mathbb{R}_{\not\geq \times \diagdown}$. 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.

4.1.2 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 previous 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 $a_i \in \mathbb{R}_{\gg \times \diagdown}$, $b_j \in \mathbb{R}_{\ltimes \times \diagdown}$, and $c_k \in \mathbb{R}_{\not\geq \times \diagdown}$. 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 $y \in \mathbb{R}_{\aleph \times \not\exists}$, $X \in \mathbb{R}_{\aleph \times \imath}$, $\beta \in \mathbb{R}_{\imath \times \not\exists}$, and
$sigma\epsilon \in \mathbb{R}_{\aleph \times \not\exists}$. 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}.

4.2 Gibbs Sampling

Following the formulation of the model, a Gibbs Sampling algorithm is used to repeatedly generate samples from the full conditional of each parameter in the model statistical model, which iteratively creates 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.

References

Angel, E. (2000). *Interactive computer graphics : A top-down approach with opengl*. Boston, MA: Addison Wesley Longman.

Angel, E. (2001a). *Batch-file computer graphics : A bottom-up approach with quicktime*. Boston, MA: Wesley Addison Longman.

Angel, E. (2001b). *Test second book by angel*. Boston, MA: Wesley Addison Longman.