# Assignment #6 – Machine Learning – Professor Haugh

## **Jaime Gacitua**

## **jg3499**

## **Due Wednesday April 06, 2016**

## Question 1

Demonstrating by induction,

Base case:

The optimization model is,

The Lagrangean is,

Differentiating and equaling to zero, to find the maximizer,

This is the equation that solves for the Eigen-value and Eigen-vector, therefore for the base case, the proposed model solves for.

Base case:

The optimization model corresponds to

The Lagrangean is,

Differentiating and equaling to zero,

## Question 2

1. Optimizing Z for fixed B, the model to optimize is

Calculating the first order conditions,

Having constant, for each, we have a system of equations, defined by the vector of equations above.

It is not a problem is the system of equations is under-determined (there are less observations in the nth data column of X than there are components of B). We can use the pseudo-inverse of the matrix to provide a minimal solution.

1. Optimizing for B for fixed Z, the first order conditions yield,

Having constant, for each, we have a system of equations, defined by the vector of equations above.

1. To code the function, I will use the following logic to construct the systems of equations.

For the first step, create systems of equations, of size M, of the form, where

For the second step, create systems of equations of the form, where

The convergence criteria will be based on parameter. Let represent the elements of Z and B on iteration t.

We will stop iterating when,

## Question 3

1. To make the problem manageable, I decided to use users, movies, and will apply the formulas with total number of PCA dimensions.

The data matrix, has shape: the users is the dimensionality of the dataset, and n as number of observations.

The initial basis was chosen as

The tolerance criteria was set at. The tolerance indicator is detailed in question 2.(c).

The script took to run. **The RMSE is 24549.7**

The output of the script was the following:

[1] "Starting Iterations"

[1] "-----------"

Iteration = 1

Tolerance = 1

Convergence = 2900.13

RMSE = 24577.08

Iteration Time = 30.21 0 30.26 NA NA

Total Time = 30.21 0 30.26 NA NA

[1] "-----------"

Iteration = 2

Tolerance = 1

Convergence = 22099.94

RMSE = 24567.16

Iteration Time = 27.4 0 27.47 NA NA

Total Time = 57.61 0 57.73 NA NA

[1] "-----------"

Iteration = 3

Tolerance = 1

Convergence = 1407.341

RMSE = 24558.37

Iteration Time = 30.2 0 30.36 NA NA

Total Time = 87.81 0 88.09 NA NA

[1] "-----------"

… Iteration 4 to 36…

Iteration = 37

Tolerance = 1

Convergence = 1.296265

RMSE = 24549.54

Iteration Time = 32.06 0 32.17 NA NA

Total Time = 1116.56 0.24 1120.51 NA NA

[1] "-----------"

Iteration = 38

Tolerance = 1

Convergence = 1.13819

RMSE = 24549.62

Iteration Time = 29.52 0 29.58 NA NA

Total Time = 1146.08 0.24 1150.09 NA NA

[1] "-----------"

Iteration = 39

Tolerance = 1

Convergence = 0.9894758

RMSE = 24549.7

Iteration Time = 22.34 0 22.34 NA NA

Total Time = 1168.42 0.24 1172.43 NA NA

1. I ran the code with different starting points. We can clearly reach better local minima if we try different starting points, therefore this must be of recurring practice in the industry. On the other hand, I see that this is a very intensive task from a computational standpoint.

This process took is around 5 hours of computer time.

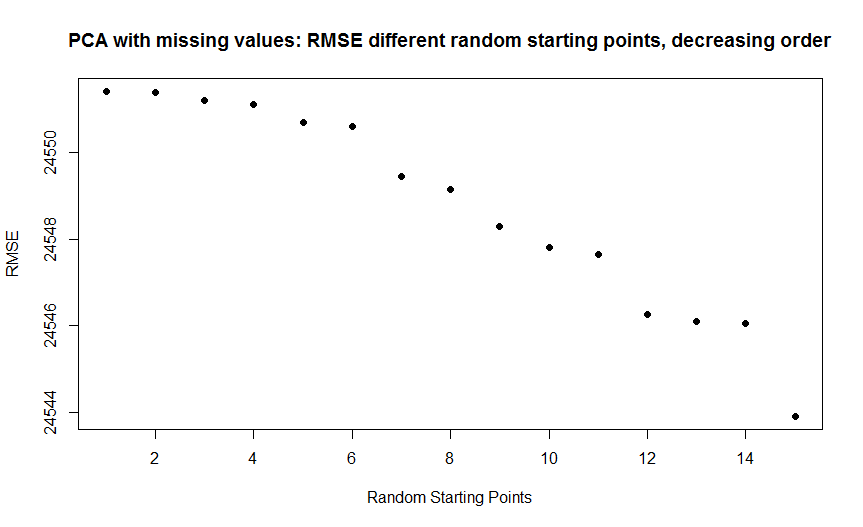
I generated the random initial basis drawing each element of the matrix from a standard normal distribution.

The RMSE obtained are in the following list, and plotted below.

> RMSE

[1] 24551.40 24549.45 24551.20 24550.71 24546.27 24551.41 24549.14 24551.11 24550.60 24547.82 24546.10 24543.92 24547.64

[14] 24546.05 24548.29



1. Using this code,

The output matrix is an approximation for the first columns of the matrix of factor loadings, in the lecture notes represented with the Greek Letter.

The output matrix is an estimated reconstruction of each data point in the PCA space.

To estimate the full matrix of ratings of, which we will call , we can do

If we want to look at movies to recommend to user,

1. Extract the row of the matrix, represented as This is a row vector of dimension n.
2. We create a new row vector , containing only the unrated movies. We would calculate it with pair-wise multiplication, for each movie as follows,
3. Within this subset, recommend the movies with the highest rating.

**If we want to recommend a movie of a specific genre, using the definition of matrix, add two steps:**

1. If we want to recommend a specific genre to user, we get the column vector , and make pairwise multiplication,

Recall that is the set of movies.

1. The final vector represents all the recommended movies from genre, for user.

## Question 4

1. I wrote the code that can be found in the appendix. The function I wrote is callable through the following syntax:

pageRank(connections, epsilon, start.vector, convergence)

* Connections: square matrix of length d, containing 0’s and 1’s. The 1’s indicate that webpage of row has a link to page of column.
* Epsilon: scalar value that allows to have an irreducible matrix
* Start.vector: Row vector of size d, to stat off the iterations.
* Convergence: stopping criteria for the iterations

1. The resulting vector with is displayed below. The output makes a lot of sense, for example:

* Page 3 has the highest rank, and it makes sense because it has the highest amount of inbound links.
* Page 1 has a very high rank, similar to Page 3, because it is basically the only page leading out of 3.
* Pages 4 and 6 have the lowest rank, and we can see that they have no inbound links.

> pi

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 0.3538667 0.1860446 0.3744637 0.025 0.035625 0.025

1. The impact of is extremely relevant. I solved the model for values of ranging from 0.1 to 1, in steps of 0.5.

The resulting vectors are represented below, as numbers, and as colors.

* When epsilon is close to zero, the webpage links are relevant.
* The closer we get to, the values for the limiting probabilities are more similar to each other.
* At, all of the pages have the same rank.

> pi.matrix

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 0.3690412 0.1902649 0.3831939 0.01666667 0.02416667 0.01666667

[2,] 0.3538667 0.1860446 0.3744637 0.02500000 0.03562500 0.02500000

[3,] 0.3390041 0.1822408 0.3654218 0.03333333 0.04666667 0.03333333

[4,] 0.3243365 0.1789440 0.3560946 0.04166667 0.05729167 0.04166667

[5,] 0.3100062 0.1760022 0.3464916 0.05000000 0.06750000 0.05000000

[6,] 0.2960187 0.1735199 0.3365031 0.05833333 0.07729167 0.05833333

[7,] 0.2823941 0.1713934 0.3262125 0.06666667 0.08666667 0.06666667

[8,] 0.2691870 0.1696405 0.3155475 0.07500000 0.09562500 0.07500000

[9,] 0.2564290 0.1682739 0.3044637 0.08333333 0.10416667 0.08333333

[10,] 0.2441521 0.1672259 0.2929970 0.09166667 0.11229167 0.09166667

[11,] 0.2324352 0.1664768 0.2810880 0.10000000 0.12000000 0.10000000

[12,] 0.2213324 0.1660213 0.2686879 0.10833333 0.12729167 0.10833333

[13,] 0.2108980 0.1658082 0.2557938 0.11666667 0.13416667 0.11666667

[14,] 0.2012329 0.1657715 0.2423706 0.12500000 0.14062500 0.12500000

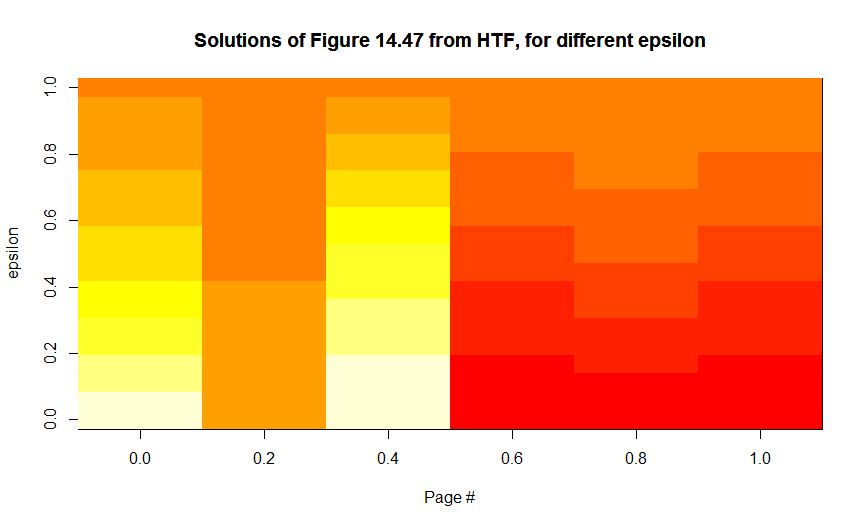
[15,] 0.1923533 0.1659000 0.2284133 0.13333333 0.14666667 0.13333333

[16,] 0.1843698 0.1661194 0.2138858 0.14166667 0.15229167 0.14166667

[17,] 0.1773750 0.1663687 0.1987562 0.15000000 0.15750000 0.15000000

[18,] 0.1714427 0.1665781 0.1830208 0.15833333 0.16229167 0.15833333

[19,] 0.1666667 0.1666667 0.1666667 0.16666667 0.16666667 0.16666667



# Appendix

## Code for Q2

PCA\_missing <- function(numBasis, data, data.avail, tolerance, startB){

library("corpcor")

# On the data matrix, the rows are the features, and columns are the observations.

# Get n = columns of X (number of data points)

n = ncol(data)

# Get d = rows of x (number of features, dimension of x)

d = nrow(data)

# Initialize matrices

Z <- mat.or.vec(numBasis,n)

B <- mat.or.vec(d,numBasis)

B <- startB

cat('Starting Iterations\n')

convergence <- 1000000

iter <- 1

ptm <- 0

ttm <- proc.time()

while (convergence > tolerance) {

ptm <- proc.time()

cat("-----------\n")

cat('Iteration = ', iter, "\n")

## First Step, solve i = 1,...,n systems of equations with B fixed. We get Z.

for (i in 1:n){

# System of equations M\_i \* z\_i = c\_i

# Define the matrix M\_i

M <- mat.or.vec(numBasis, numBasis)

for (k in 1:numBasis){

for (l in 1:numBasis){

# Sum over dimension of x

for (j in 1:d){

M[k,l] <- M[k,l] + data.avail[j,i]\*B[j,l]\*B[j,k]

}

}

}

# Define the vector c\_i

c <- mat.or.vec(numBasis,1)

# Sum over dimension of x

for (l in 1:numBasis){

c[l] = 0

# Sum over dimension of x

for (j in 1:d){

if (data.avail[j,i] == 1) {

c[l] <- c[l] + data.avail[j,i]\*data[j,i]\*B[j,l]

}

}

}

# Save z.old

Z.old <- Z

# Solve for z\_i

# print("M = ")

# print(M)

# cat("c = ", c, "\n")

Z[,i] <- pseudoinverse(M) %\*% c

# print("Z = ")

# print(Z)

}

## Second Step, solve j = 1,...,d systems of equations with Z fixed. We get B.

for (j in 1:d){

# System of equations F\_j \* b\_j = m\_j

# Define the matrix F\_j

Fmat <- mat.or.vec(numBasis, numBasis)

for (k in 1:numBasis){

for (l in 1:numBasis){

# Sum over dimension of x

for (i in 1:n){

Fmat[k,l] <- Fmat[k,l] + data.avail[j,i]\*Z[l,i]\*Z[k,i]

}

}

}

# Define the vector c\_i

m <- mat.or.vec(numBasis,1)

# Sum over dimension of x

for (l in 1:numBasis){

# Sum over dimension of x

for (i in 1:n){

if(data.avail[j,i] == 1){

m[l] <- m[l] + data.avail[j,i]\*data[j,i]\*Z[l,i]

}

}

}

# Save B.old

B.old <- B

# Solve for b\_j

# B[,j] <- solve(Fmat, m)

B[j,] <- pseudoinverse(Fmat) %\*% m

# print("B = ")

# print(Z)

}

# Calculate convergence criteria

# Squared difference between iteration results

convergence <- 0

for(l in 1:numBasis) {

for(i in 1:n) {

convergence <- convergence + (Z[l,n]-Z.old[l,n])^2

}

for(j in 1:d) {

convergence <- convergence + (B[j,l]-B.old[j,l])^2

}

}

iter <- iter+1

### Calculate Root-mean-squared-error (RMSE)

# RMSE Compute objective function

RMSE <- 0

temp.RMSE <- 0

for(i in 1:n){

for(j in 1:d){

if(data.avail[j,i] == 1){

for(k in 1:numBasis){

temp.RMSE <- temp.RMSE + B[j,k]\*Z[k,i]

}

RMSE <- RMSE + (data[j,i]-temp.RMSE)^2

}

}

}

# RMSE Count number of observations, and divide

data.avail.count <- 0

for(i in 1:n){

for(j in 1:d){

data.avail.count <- data.avail.count + data.avail[j,i]

}

}

# RMSE Divide and apply square root

RMSE <- RMSE / data.avail.count

RMSE <- RMSE^(1/2)

### Output

# Convergence Measures

cat('Tolerance =\t', tolerance, "\n")

cat('Convergence =\t', convergence, "\n")

cat('RMSE =\t\t', RMSE, "\n")

# Timing

cat("Iteration Time =\t", proc.time() - ptm, "\n")

cat("Total Time =\t\t", proc.time() - ttm, "\n\n")

}

returnList <- list("Convergence" = convergence, "RMSE" = RMSE, "Total Time" = proc.time() - ttm,

"Basis" = B)

return(returnList)

}

## Code for Q3

source("Q2.R") # Function PCA with missing data.

# Input Parameters

num.users = 200 # Max 943

num.movies = 400 # Max 1682

num.pca.basis = 5

convergence = 1

### Prepare the database (snippet from Q2 HW5)

movies <- read.table("u.item", sep = "|", header = FALSE, stringsAsFactors = FALSE, quote="")

movies <- movies[,c(1,2)]

names(movies) <- c("movieid","movie")

rank <- read.table("u.data", sep = "\t", header = FALSE, stringsAsFactors = FALSE,

col.names = c("userid","movieid","rating","ts"))

critics <- merge(movies, rank, by = "movieid")

critics$movie <- NULL

critics$ts <- NULL

names(critics) <- c("movieid","person","rank")

### Prepare matrices

# Data matrix

data = matrix(nrow = num.users, ncol = num.movies)

gamma = matrix(nrow = num.users, ncol = num.movies)

for(i in 1:nrow(critics)) {

if (critics[i,2] <= num.users & critics[i,1] <= num.movies) {

data[critics[i,2] , critics[i,1]] <- critics[i,3]

}

}

# Gamma matrix

for(i in 1:ncol(data)){

for(d in 1:nrow(data)){

if(is.na(data[d,i]) ){

gamma[d,i] = 0

} else {

gamma[d,i] = 1

}

}

}

# Starting Basis

start.pca.basis = matrix(0, nrow = nrow(data), ncol = num.pca.basis)

# Part A

for(d in 1:nrow(data)){

start.pca.basis[d,d %% num.pca.basis + 1] = 1

}

finalB <- PCA\_missing(num.pca.basis, data, gamma, convergence, start.pca.basis)

finalB

# Part B

K = 15

RMSE <- mat.or.vec(K,1)

iter = 1

for(iter in 1:K){

start.pca.basis <- matrix( rnorm(nrow(data)\*num.pca.basis,mean=0,sd=1), nrow(data), num.pca.basis)

output <- PCA\_missing(num.pca.basis, data, gamma, convergence, start.pca.basis)

RMSE[iter] <- output$RMSE

}

RMSE

plot(RMSE)

RMSE.sorted <- sort(RMSE, decreasing = TRUE)

plot(RMSE.sorted, pch = 19, xlab = "Random Starting Points", ylab = "RMSE",

main = "PCA with missing values: RMSE different random starting points, decreasing order")

## Code for Q4

pageRank <- function(connections, epsilon, start.vector, convergence){

d <- nrow(connections)

## Create transition matrix as Q\_jk = 1/c(j)

## Where c(j) is the number of outbout links from page i

# Initialize transition matrix as zeros.

trans.mat <- matrix(0, nrow = d, ncol = d)

# Compute c(j)

c <- rowSums(connections)

# Populate transition matrix

for(j in 1:d){

for(k in 1:d){

if (connections[j,k] == 1){

trans.mat[j,k] <- 1 / c[j]

}

}

}

# Create irreductible transition matrix by adding epsilon to all elements

ones <- matrix(data = 1, nrow = d, ncol = d)

trans.mat.irr <- (1-epsilon) \* trans.mat + epsilon/d\*ones

# Initialize the iterating vector

mu = start.vector

mu.old = start.vector

check <- 10000

iter <- 1

while(check > convergence){

# Update mu

mu <- mu.old %\*% trans.mat.irr

# Calculate convergence criteria

check <- dist(rbind(mu,mu.old))

# Record old mu

mu.old <- mu

# Output stats

cat('Done with iteration:', iter, '\n')

cat('Convergence:', check, ' / ', convergence, '\n')

iter <- iter + 1

}

return(mu)

}

connections <- matrix(c(0,0,1,1,0,0,

1,0,0,0,0,1,

1,1,0,1,1,0,

0,0,0,0,0,0,

0,0,0,0,0,1,

0,0,0,0,0,0),

nrow = 6, ncol = 6)

initial <- matrix(1/nrow(connections), nrow = 1, ncol = nrow(connections))

epsilon <- 0.15

convergence <- 0.0001

# Question b

pi <- pageRank(connections,epsilon,initial,convergence)

pi

# Question c

epsilon.values <- matrix(seq(0.1, 1, 0.05), nrow = 19)

pi.matrix <- matrix(0, nrow = nrow(epsilon.values), ncol = nrow(connections))

for(iter in 1:nrow(epsilon.values)){

pi.matrix[iter,] <- pageRank(connections,epsilon.values[iter,1],initial,convergence)

}

pi.matrix

a <-image(t(pi.matrix))

title(main = 'Solutions of Figure 14.47 from HTF, for different epsilon',

xlab = 'Page #', ylab = 'epsilon')