ISE 426 Optimization

Final Project

Movie Stars

Yuheng Huang, Puxin Xu

Problem Description:

Given a matrix which contains 24 ratings from 5 users to 5 movies, give a prediction of the missing value in the matrix.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Movie 1 | Movie 2 | Movie 3 | Movie 4 | Movie 5 |
| User 1 | 4 | 5 | 1 | 4 | 4 |
| User 2 | 4 | 1 | 5 | 5 | 2 |
| User 3 | 2 | 4 | 2 | 2 | 4 |
| User 4 | 2 | 3 | 4 | 5 | 2 |
| User 5 | 5 | 3 | 1 | 2 | ? |

Assumption:

*Assumption 1:*

The rating from user *i* to movie *j* is function of the user’s preference, movie’s quality, and the features of the movie:

*Assumption 2:*

The actual rating s slightly different from true value of, because of all kinds of minor factors:

We also assume is independent from each other and normally distributed with 0 mean:

0,

Models:

We try to find such a dense matrix that is closed to the true value of the given sparse rating matrix.

From the distribution of we know least square regression gives the likelihood maximum estimation. So our primary goal is to:

O denotes the observed pairs of (i,j).

There are two ways of achieving this goal:

##### Model 1:

U is the users’ preference matrix, and M is the movies’ features matrix. Thus the rating matrix R is:

R=MTU

=

=

Our objective is to find such two matrixes U and M that minimize the cost function with L-2 regulation.

##### Features:

For a size-24 training set, maximum number of features is given by:

In general, for to get a meaningful result, the upper bound of number of features is given by:

##### Training Algorithm:

This is a non-convex quadratic problem because of. Theoretically, gradient descent gives approximate approach but when the matrix gets bigger gradient descent is slow and expensive. Another way of doing this is alternating least square.

The gradient vector of the cost function J:

In iteration, fix M and update:

Then fix U and update

Repeat until converge.

##### Model 2:

Complete the sparse matrix while avoiding adding information in the matrix:

In practice, the relaxed problem is:

Implement SVD on matrix R:

According to SVD,

,

To solve avoid meaningless computation on unobserved entries, define a matrix (with dimension

which is a projection of the matrix onto the observed entries. In the same spirit, define the complementary projection via , rewrite as.

Now, we rewrite the objective as:

Y is the prediction and R is the observed matrix.

##### Training Algorithm:

1. Initialize
2. Repeat{
3. Compute
4. If exit.
5. Assign }
6. Assign

Results of the original problem:

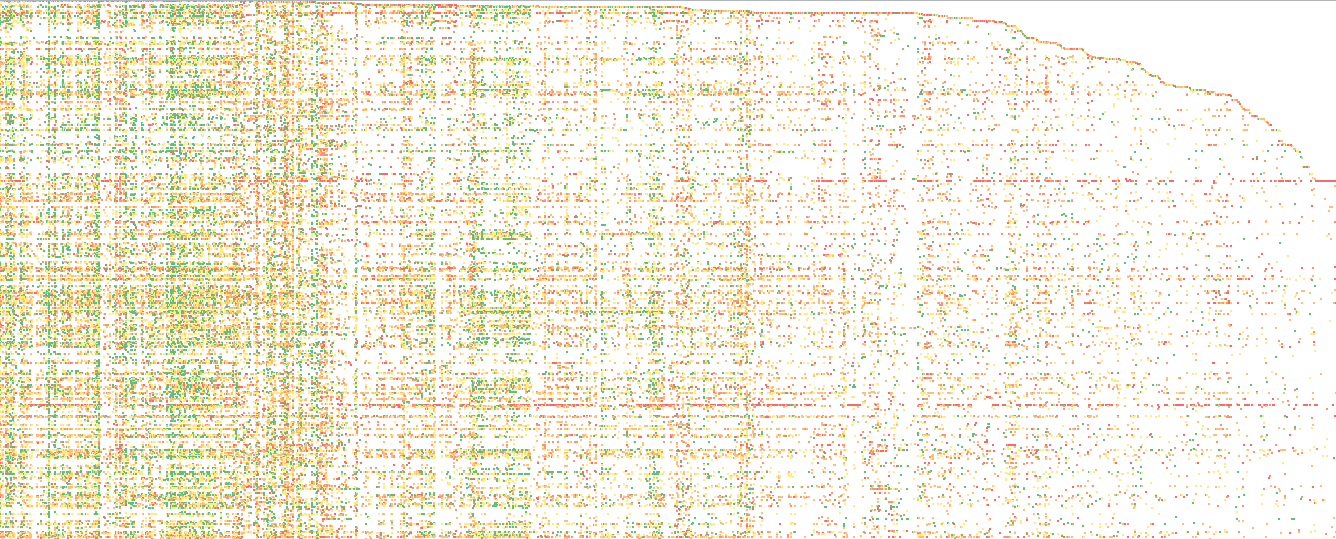
* **Using SVD, the grade is:**
* **Using ALS, the grade is:**

Results on larger data set:

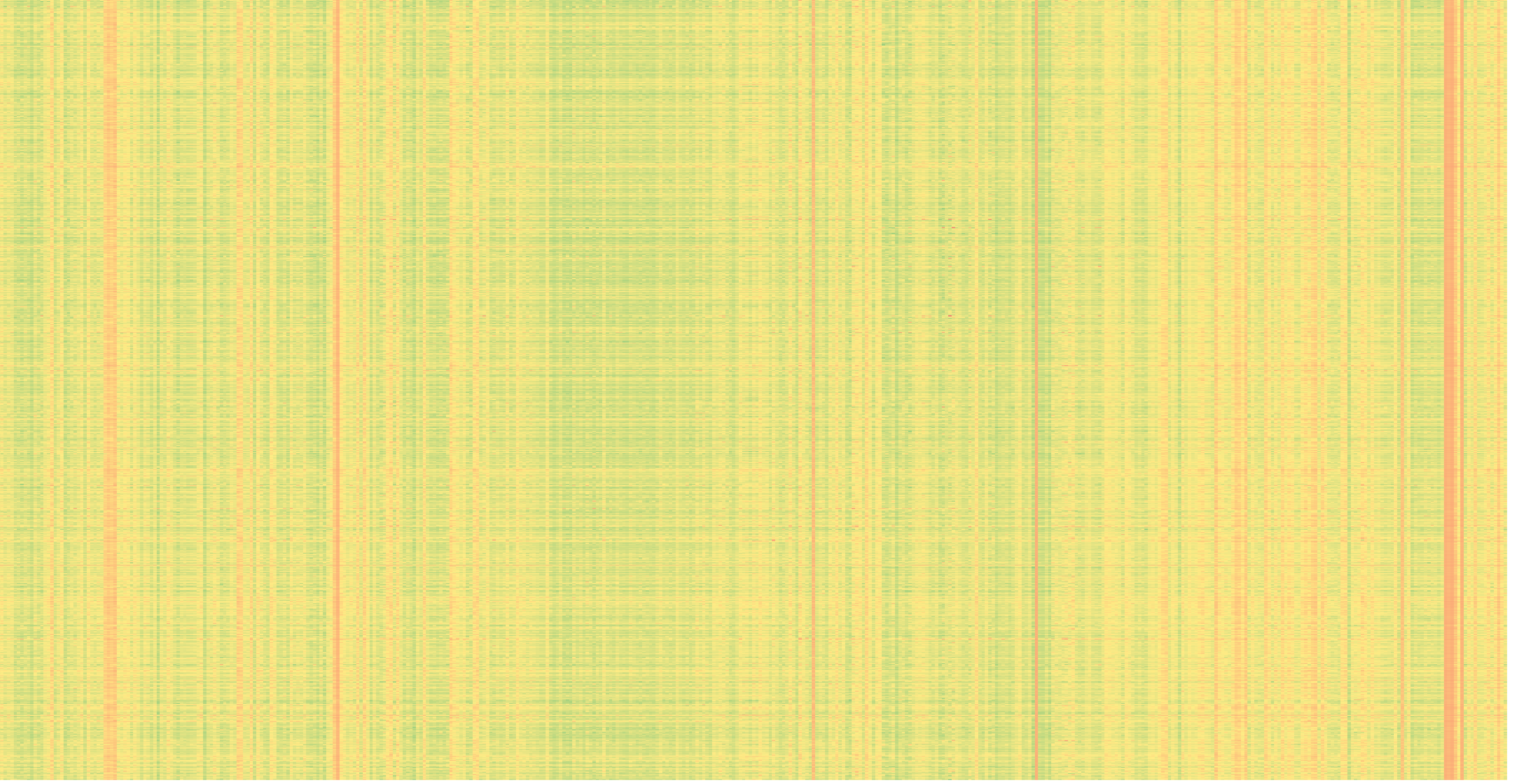
We also ran our model on a data set provided by MovieLenens (<http://grouplens.org/datasets/movielens/>), which contains 100,000 ratings from 1000 users on 1700 movies.

Two models give slightly different results. Here are heat-maps of the original data followed by two model’s predictions:

(Green spots mean high ratings, red ones mean low ratings and yellow ones are in between.)



*Figure 1.Movie Rating Distribution (Original data set)*



*Figure 2.Movie Rating Distribution (Prediction by Model 1)*



*Figure 3.Movie Rating Distribution (Prediction by Model 2)*

# Reference:

Donatelli, M., Neuman, A., & Reichel, L. (2012). Square regularization matrices for large linear discrete ill-posed problems. Numer. Linear Algebra Appl. Numerical Linear Algebra with Applications, 896-913.

Koren, Y. (2009). The BellKor Solution to the Netflix Grand Prize.

http://www.netflixprize.com/assets/GrandPrize2009\_BPC\_BellKor.pdf

Koren, Y., Bell, R., & Volinsky, C. (n.d.). Matrix Factorization Techniques for Recommender Systems. Computer, 30-37. http://stanford.edu/~rezab/dao/notes/lec14.pdf

Appendix

Source code of model 1:

R 3.1.2:

obRec\_1 <- function(M){

x <- c();

y <- c();

for(i in 1:dim(M)[1]){

for(j in 1:dim(M)[2]){

if(!is.na(M[i,j])){

x <- c(x,i);

y <- c(y,j);

}

}

}

rM <- matrix(c(x,y),ncol = 2);

return(rM);

}

Eunorm <- function(X) {

sqrt(sum(X^2))

}

ALS <- function(R, lambda = 10, nf = 25, itemax = 1000, default = 2, ea = 1e-4){

Rec = obRec\_1(R);

W <- !is.na(R);

show(sum(W))

M <- R;

M[is.na(R)]<- 0;

np <- dim(R)[1];

nm <- dim(R)[2];

show((np+nm)\*nf);

if((np+nm)\*nf >= sum(W)){show('WARNING!: More Variables than observations!')}

Ik <- diag(nf);

X <- replicate(np, rnorm(nf)\*10);

Y <- replicate(nm, rnorm(nf)\*10);

E <- t(X)%\*%Y - R;

EZ <- E;

EZ[is.na(E)]<-0;

J <- Eunorm(EZ) #+ lambda\*(Eunorm(X)+Eunorm(Y));

for( u in 1:itemax){

#update X;

for (i in 1:np) {

YYT <- matrix(rep(0,times = nf\*nf),nrow = nf, ncol = nf);

wy<-t(matrix(rep(W[i,],times =nf),nrow = nm))

YYT <- YYT + (wy\*Y)%\*%t(wy\*Y);

A <- YYT+lambda\*Ik;

RY <- matrix(rep(0,times = nf),nrow = nf, ncol = 1);

for (j in 1:nm){

if(W[i,j] == 1){

RY <- RY + M[i,j]\*Y[,j];

}

}

b <- RY;

X[,i] <- solve(A)%\*%b;

}

show(paste("Update X ",u," times."))

#update Y;

for (j in 1:nm) {

XXT <- matrix(rep(0,times = nf\*nf),nrow = nf, ncol = nf);

wx<-t(matrix(rep(W[,j],times =nf),nrow = np))

XXT <- XXT + (wx\*X)%\*%t(wx\*X);

A <- XXT+lambda\*Ik;

RX <- matrix(rep(0,times = nf),nrow = nf, ncol = 1);

for (i in 1:np){

if(W[i,j] == 1){

RX <- RX + M[i,j]\*X[,i];

}

}

b <- RX;

Y[,j] <- solve(A)%\*%b;

}

show(paste("Update Y ",u," times."))

E <- t(X)%\*%Y - R;

EZ <- E;

EZ[is.na(E)]<-0;

J <- c(J,Eunorm(EZ) );#+ lambda\*(Eunorm(X)+Eunorm(Y))

if(J[length(J)-1]-J[length(J)]<ea){

break;

}

}

plot(J[3:length(J)]);

predict = t(X)%\*%Y;

write.csv(predict, file = "M100KPredict\_lambda10.csv")

#return(predict);

show(J[length(J)]);

}

Source code of model 2:

R 3.1.2:

# The function

Self\_SVD <- function (M, lambda = 0, maxiteN = 100, epsilon = 1e-5 ){

library(svd)

# record the all plot of reliable values

N <- matrix(NA,2,nrow(M)\*ncol(M))

t <- 1

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

for (j in 1:ncol(M)){

if (!is.na(M[i,j])){

N[1,t] <- i

N[2,t] <- j

t <- t+1}}}

N <- N[,1:t]

Itr <- c(N[1,])

Jtr <- c(N[2,])

# Change NA to 0

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

for (j in 1:ncol(M)){

if (is.na(M[i,j])) M[i,j] <- 0}}

# Algorithm

## set default Z\_old, Z\_new

Z\_old <- matrix(0, nrow(M),ncol(M))

Z\_new <- matrix(0, nrow(M),ncol(M))

## Repeat

count\_ite <- 0

for (j in 1:maxiteN) {

### Caculate Z\_new

for (i in 1:length(Itr)) Z\_old[Itr[i],Jtr[i]] <- 0

Z\_temp <- Z\_old + M

SVD\_Z\_temp <- svd(Z\_temp)

SVD\_Z\_temp$d <- max(0, SVD\_Z\_temp$d - lambda)

d <- matrix(0, ncol(SVD\_Z\_temp$u), ncol(SVD\_Z\_temp$v))

for (k in 1:length(SVD\_Z\_temp$d)) d[k,k] <- SVD\_Z\_temp$d[k]

Z\_new <- SVD\_Z\_temp$u %\*% d %\*% t(SVD\_Z\_temp$v)

### check the eps

current\_eps <- (norm(Z\_new - Z\_old, type = "F"))/(norm(Z\_old, type = "F")+0.001)

if (current\_eps < epsilon) break

### Assign

Z\_old <- Z\_new

count\_ite <- count\_ite + 1 # count iterations

}

## Assign

for (i in 1:length(Itr)) Z\_old[Itr[i],Jtr[i]] <- 0

Z <- Z\_old + M

out = list(Z,count\_ite, current\_eps)

return(out)

}

# Running

t0 <- Sys.time() # start the timer

M <- read.table("u.data")

Result <- Self\_SVD(M, lambda = 5, maxiteN = 500, epsilon = 0.1)

Z <- Result[[1]]

iteTimes <- Result[[2]]

eps <- Result[[3]]

write.csv(Z,"Z.csv")

t1 <- Sys.time() # stop the timer

running\_time <- t1 - t0

iteTimes

running\_time

eps