The Two Imputation Strategies

David Gerard

2016-03-29

Abstract

I run some simulations to see how well variational imputation does vs just inserting zeros.

Implementation Example

I generate data where 70% of the observations are missing.

```r
library(flashr)
set.seed(831)
n <- 50
p <- 50
l <- rnorm(n)
f <- rnorm(p)
Theta <- l %*% t(f)
E <- matrix(rnorm(n * p), nrow = n)
Y <- Theta + E

pmiss <- 0.7
Omega <- sample(1:(n * p), size = n * p * pmiss)
Y[Omega] <- NA

tout <- tflash(Y)
fout <- flashr:::flash(Y)
```

I then calculate the correlations between the true and estimated components using one of two imputation strategies. (1) Just inserting zeroes into for the missing values and (2) updating the missing values as parameters in the VEM algorithm.

```r
cat("Absolute Values of Correlations Between True and Estimated:\n",
    "Loadings Variational Approach:", abs(cor(tout$post_mean[[1]], l)), "\n",
    "      Loadings Zeros Approach:", abs(cor(fout$l, l)), "\n",
    " Factors Variational Approach:", abs(cor(tout$post_mean[[2]], f)), "\n",
    "       Factors Zeros Approach:", abs(cor(fout$f, f)), "\n")
```

```
## Absolute Values of Correlations Between True and Estimated:
##  Loadings Variational Approach: 0.9132
##        Loadings Zeros Approach: 0.8306
##   Factors Variational Approach: 0.922
##         Factors Zeros Approach: 0.855
```

Longer Simulation Study

I set 70% of the observations to be missing and generate a new mean each iteration where factors and loadings are from standard normals.

```r
rm(list = ls())
library(flashr)
set.seed(112)
n <- 50
p <- 50
pmiss <- 0.7

itermax <- 100
cor_mat <- matrix(NA, nrow = itermax, ncol = 4)
sse_mat <- matrix(NA, nrow = itermax, ncol = 2)
for (iter_index in 1:itermax) {
  l <- rnorm(n)
  f <- rnorm(p)
  Theta <- l %*% t(f)
  E <- matrix(rnorm(n * p), nrow = n)
  Y <- Theta + E

  Omega <- sample(1:(n * p), size = n * p * pmiss)
  Y[Omega] <- NA

  tout <- tflash(Y)
  fout <- flashr:::flash(Y)


  sse_mat[iter_index, 1] <- sum((form_outer(tout$post_mean) - Theta)^2)
  sse_mat[iter_index, 2] <- sum((fout$l %*% t(fout$f) - Theta)^2)

  cor_mat[iter_index, 1] <- abs(cor(tout$post_mean[[1]], l))
  cor_mat[iter_index, 2] <- abs(cor(tout$post_mean[[2]], f))
  cor_mat[iter_index, 3] <- abs(cor(fout$l, l))
  cor_mat[iter_index, 4] <- abs(cor(fout$f, f))
}
```

Using the VEM to update the missing values performs better than just inserting zeros.

```r
cat("   Mean Sum of Squared Errors VEM:", mean(sse_mat[, 1]), "\n",
    "Mean Sum of Squared Errors Zeros:", mean(sse_mat[, 2]))
```

```
##    Mean Sum of Squared Errors VEM: 1415
##  Mean Sum of Squared Errors Zeros: 1720
```

```r
cat("   Mean Cor Loadings VEM:", mean(cor_mat[, 1], na.rm = TRUE), "\n",
    "Mean Cor Loadings Zeros:", mean(cor_mat[, 3], na.rm = TRUE), "\n",
    "   Mean Cor Factors VEM:", mean(cor_mat[, 2], na.rm = TRUE), "\n",
    " Mean Cor Factors Zeros:", mean(cor_mat[, 4], na.rm = TRUE))
```

```
##    Mean Cor Loadings VEM: 0.8235
##  Mean Cor Loadings Zeros: 0.7536
##     Mean Cor Factors VEM: 0.8381
##   Mean Cor Factors Zeros: 0.7708
```

I don't know if it's a big deal, but my code for some reason shrinks the factors toward zero more often. For example, in 29 percent of these trials versus 0 percent of the trials when I used `flash`.