Simulation2

September 25, 2019

1 GD on Manifold

Let $\mathcal{G}(n,r)$ denotes the Grassmann manifold — the collection of r-dimensional linear subspaces of the n-dimensional space. More specifically, any point in $\mathcal{G}(n,r)$ is an equivalent class of a $n \times r$ orthonormal matrix.

Unlike previous methods, here we try to use SVD to estimate the complete matrix. That is, minimize the loss function as follows

$$\text{minimize}\, F(X,Y) \qquad \text{s.t.} \quad X \in \mathcal{G}(n_1, r), Y \in \mathcal{G}(n_2, r),$$

where

$$F(X,Y) := \frac{1}{2} \min_{S \in \mathbb{R}^{r \times r}} \left\| \mathcal{P}_\Omega \left(M^* - XSY^T \right) \right\|_\mathrm{F}^2 .$$

Taking gradients over the Grassmann manifold (Keshavan, R. H., Montanari, A., & Oh, S. (2009). Matrix Completion from a Few Entries. Retrieved from http://arxiv.org/abs/0901.3150) yields

$$\begin{aligned} \nabla F_X(X,Y) &= \left(I - XX^T\right) P_\Omega \left(XSY^T - M\right) YS^T, \\ \nabla F_Y(X,Y) &= \left(I - YY^T\right) P_\Omega \left(XSY^T - M\right)^T XS. \end{aligned}$$

Let $-\nabla F_X(X,Y) = U_t D_t V_t^T$ be its compact SVD, then the geodesic on the manifold along the gradient direction is given by

$$X_t(\eta_t) = \left[X_t V_t \cos\left(D_t \eta_t\right) + U_t \sin\left(D_t \eta_t\right)\right] V_t^T.$$

A similar expression holds for $Y(t)$.

1.1 Step Size Selection

Following the step selection procedure in [Samelson, H., Hobby, C. R., Dugundji, J., & Arens, R. (1971). Pacific journal of mathematics. Pacific Journal of Mathematics, 37(3), 1], the step size $\eta_t = 1/2^{m-1}$ where m is the smallest positive integer such that

$$f(X_t(\eta_t)) - f\left(X_k\right) \le -\frac{1}{2} \eta_t \left\| \nabla f\left(X_k\right) \right\|_F^2 .$$

1.2 Optimization over S

Setting to zero the derivative of $F(\cdot)$ w.r.t. S yields

$$X^T \mathcal{P}_\Omega (XSY^T) Y = X^T \mathcal{P}_\Omega (M) Y.$$

Since the above equation has no analytical solution, we try to solve it using vanilla GD. That is, each evaluation of F is an optimization problem.

1.3 Initialization

- Trimming Set a row (or column) to 0 if it contains more than $2|E|/n_1$ (or $2|E|/n_2$) revealed entries. It is claimed that trimming helps recovering the rank-r structure.
- rank-r Projection Use the rSVD to obtain a rank-r approximation of the trimmed matrix. It is also used as the initialization of X and Y.

```r
# Generate Data -----------------------------------------------------------

# L: n1*r, R: n2*r, M: n1*n2
n1 = 10; r = 5; n2 = 10
missfrac = 0.1

set.seed(1983)
L_true = matrix(rnorm(n1*r), n1, r)
set.seed(831)
R_true = matrix(rnorm(n2*r), n2, r)
M_true = L_true %*% t(R_true)


Projector = function(M_complete, miss_id = imiss)
{
  M_miss = M_complete
  M_miss[miss_id] = 0
  return(M_miss)
}


set.seed(19)
imiss = sample(seq(n1*n2), n1*n2*missfrac, replace = F)
M_obs = Projector(M_true, imiss)



# GD on Manifold ----------------------------------------------------------

OptS = function(M, X, Y, S0, lr, subMaxIter){
  St = S0
  for (t in 1:subMaxIter) {
    St = St - lr * t(X) %*% Projector(X%*%St%*%t(Y)-M) %*% Y
  }
  return(St)
}


F_manifold = function(M, X, Y, S0, lr, subMaxIter){
  # S0 = 1/(1-missfrac) * t(X) %*% M %*% Y ######## Improve?
  S = OptS(M, X, Y, S0, lr, subMaxIter)
  return(1/2 * norm(Projector(M-X%*%S%*%t(Y)), 'f')^2)
}


OptSpace = function(M, X0, Y0, S0, MaxIter, sublr, subMaxIter){
  Xt = X0; Yt = Y0; St = S0
```

```r
  n1 = dim(M)[1]; n2 = dim(M)[2];
  for (i in 1:MaxIter) {
    # St = 1/(1-missfrac) * t(Xt) %*% M %*% (Yt) ###### Improve?
    St = OptS(M, Xt, Yt, St, sublr, subMaxIter)
    gradX = (diag(1, n1)-Xt%*%t(Xt)) %*% Projector(Xt%*%St%*%t(Yt)-M) %*% Yt %*% t(St)
    gradY = (diag(1, n2)-Yt%*%t(Yt)) %*% t(Projector(Xt%*%St%*%t(Yt))-M) %*% Xt %*% St
    SVD_gradX = rsvd(-gradX, r); U_gradX = SVD_gradX$u; V_gradX = SVD_gradX$v; d_gradX = SVD_gradX$d
    SVD_gradY = rsvd(-gradY, r); U_gradY = SVD_gradY$u; V_gradY = SVD_gradY$v; d_gradY = SVD_gradY$d
    F_XtYt = F_manifold(M, Xt, Yt, St, sublr, subMaxIter)
    grad_norm_sq = norm(gradX, 'f')^2 + norm(gradY, 'f')^2

    # lr = 0.001
    # X = Xt %*% V_gradX %*% diag(cos((d_gradX)*lr)) %*% t(V_gradX) + U_gradX %*% diag(sin((d_gradX)*lr
    # Y = Yt %*% V_gradY %*% diag(cos((d_gradY)*lr)) %*% t(V_gradY) + U_gradY %*% diag(sin((d_gradY)*lr

    m = 1
    alpha = 1
    while (m<=12) {
      lr = alpha / 2^(m-1)
      X = Xt %*% V_gradX %*% diag(cos((d_gradX)*lr)) %*% t(V_gradX) + U_gradX %*% diag(sin((d_gradX)*lr
      Y = Yt %*% V_gradY %*% diag(cos((d_gradY)*lr)) %*% t(V_gradY) + U_gradY %*% diag(sin((d_gradY)*lr
      if ((F_manifold(M, X, Y, St, lr, subMaxIter) - F_XtYt) < (-0.5*lr*(grad_norm_sq))) {
        break
      }
      m = m+1
    }
    Xt = X; Yt = Y
  }
  return(list(Xt = Xt, Yt = Yt))
}

library(rsvd)
Init_OptSpace = function(M){
  n1 = dim(M)[1]; n2 = dim(M)[2]
  miss_position = matrix(0, n1, n2); miss_position[imiss] = 1
  row_degree = rowSums(miss_position)
  col_degree = colSums(miss_position)
  for (j in 1:n2) {
    if (col_degree[j]>2*length(imiss)/n2) M[,j] = 0
  }
  for (i in 1:n1) {
    if (row_degree[i]>2*length(imiss)/n1) {
      M[i,] = 0
    }
  }
  rsvd_TrM = rsvd(1/(1-missfrac)*M, r)
  return(list(X0 = rsvd_TrM$u, Y0 = rsvd_TrM$v, S0 = diag(rsvd_TrM$d)))
}

init_optspace = Init_OptSpace(M_obs)
X0 = init_optspace$X0; Y0 = init_optspace$Y0; S0 = init_optspace$S0
GD_OnManifold = OptSpace(M_obs, X0, Y0, S0, 1000, 0.01, 100)
X_est = GD_OnManifold$Xt; Y_est = GD_OnManifold$Yt
```

```r
# S_est = 1/(1-missfrac) * t(X_est) %*% M_obs %*% (Y_est)
S_est = OptS(M_obs, X_est, Y_est, S0, 0.01, 1000)
M_est = X_est %*% S_est %*% t(Y_est)


norm(Projector(M_est-M_obs), 'f')/norm(M_obs, 'f')
```

```
## [1] 0.005449788
```

```r
norm(M_est-M_true, 'f')/norm(M_true, 'f')
```

```
## [1] 0.00998584
```

2 AltMin

Iterates between the following two steps using GD.

$$
\begin{aligned}
R_t &= \underset{R \in \mathbb{R}^{n_2 \times r}}{\operatorname{argmin}} \left\| \mathcal{P}_\Omega \left(L_{t-1} R^\top - M^* \right) \right\|_{\mathrm{F}}^2, \\
L_t &= \underset{L \in \mathbb{R}^{n_1 \times r}}{\operatorname{argmin}} \left\| \mathcal{P}_\Omega \left(L R_t^\top - M^* \right) \right\|_{\mathrm{F}}^2.
\end{aligned}
$$

Setting gradients to zero leads to

$$
L^T \mathcal{P}_\Omega(M) = L^T \mathcal{P}_\Omega(LR^T).
$$

The above equation does not have analytical solution, thus we adopt GD for the optimization problems.

```r
# AltMin ------------------------------------------------------------------

OptR = function(M, L, R0, lr, MaxIter){
  Rt = R0
  for (i in 1:MaxIter) {
    Rt = Rt - lr * t(Projector(L%*%t(Rt)-M)) %*% L
  }
  return(Rt)
}

OptL = function(M, L0, R, lr, MaxIter){
  Lt = L0
  for (i in 1:MaxIter) {
    Lt = Lt - lr * Projector(Lt%*%t(R)-M) %*% R
  }
  return(Lt)
}


AltMin = function(M, L0, R0, lr, MaxIter, subMaxIter){
  Lt = L0; Rt = R0
  for (t in 1:MaxIter) {
    Lt = OptL(M, Lt, Rt, lr, subMaxIter)
    Rt = OptR(M, Lt, Rt, lr, subMaxIter)
  }
  return(list(L_est = Lt, R_est = Rt))
}
```

```r
L0 = LR_decomp(M_obs/(1-missfrac))$L0; R0 = LR_decomp(M_obs/(1-missfrac))$R0
AltMin_est = AltMin(M_obs, L0, R0, 0.01, 100, 100)
L_est = AltMin_est$L_est; R_est = AltMin_est$R_est
M_est = L_est %*% t(R_est)


norm(Projector(M_est-M_obs), 'f')/norm(M_obs, 'f')
```

```
## [1] 1.093525e-06
```

```r
norm(M_est-M_true, 'f')/norm(M_true, 'f')
```

```
## [1] 8.963028e-06
```

3 ADMM