E4540_HW1

Guojing Wu, UNI: gw2383

2/23/2019

Problem 1

Step 1. Prepare the data

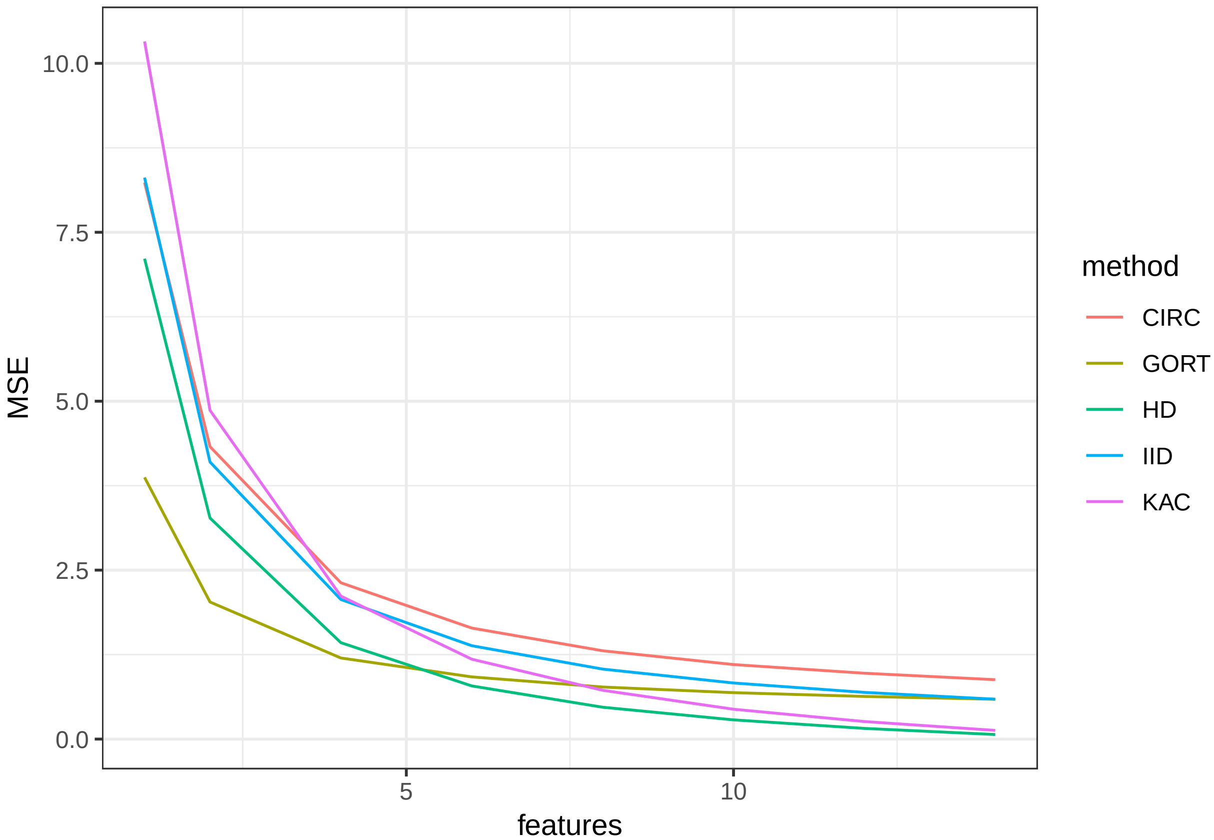
```
np <- import("numpy")
npfile <- np$load("boston-full.npz")
data.train <- npfile$f[["Xtrain"]]
data.test <- npfile$f[["Xtest"]]
data.train[,14] = 1
```

Step 2. Simulation

Calculate the average MSE of 1000 simulations. For each simulation, randomly pick 100 vector pairs to calculate the MSE (1 from Xtrain, 1 from Xtest) while make sure $|x^T y| > 1$.

```
set.seed(1)
vlen = 16
n.simu = 1000
n.pair = 100
n.feature = c(1, 2, 4, 6, 8, 10, 12, 14)
ran_dot <- function(k, G, v1, v2) {
  n.rows = sample(c(1:vlen), n.feature[k], replace = F) # randomly pick m features
  return((v1 %*% v2 - t((G[n.rows,] * (1/sqrt(n.feature[k]))) %*% t(t(v1))) %*% ((G[n.rows,] * (1/sqrt(
  }
res = tibble(IID = rep(0, length(n.feature)),
             CIRC = rep(0, length(n.feature)),
             GORT = rep(0, length(n.feature)),
             HD = rep(0, length(n.feature)),
             KAC = rep(0, length(n.feature)))

for (i in 1:n.simu) {
  # unstructured Gaussian matrices
  IID = matrix(rnorm(vlen*vlen), nrow = vlen, ncol = vlen)

  # circulant Gaussian matrices
  CIRC = matrix(nrow = vlen, ncol = vlen)
  tmp = rnorm(vlen)
  for (m in 1:vlen) CIRC[m, ] = c(tmp[m:vlen], tmp[-(m:vlen)])

  # Gaussian orthogonal matrices
  GORT = matrix(rnorm(vlen*vlen), nrow = vlen, ncol = vlen)
  for (m in 2:nrow(GORT)) {
    tmp = GORT[m,]
    for (n in 1:(m - 1)) {
      GORT[m,] = GORT[m,] - project(tmp ~ GORT[n,], coefficients = F)
    }
```

```
  }

  # random Hadamard matrices with 3 HD blocks
  had = hadamard(vlen)
  HD = had %*% diag(sample(c(-1, 1), vlen, replace = T)) %*% had %*% diag(sample(c(-1, 1), vlen, replac

  # Kac's random walk matrices given random rotation
  KAC = diag(rep(1,vlen))
  for (m in 1:vlen) {
    istar = sample(c(1:vlen), 1)
    jstar = sample(c(1:vlen), 1)
    theta = runif(1, min = 0, max = 2*pi)
    G = diag(rep(1,vlen))

    G[istar, istar] = cos(theta)
    G[istar, jstar] = sin(theta)
    G[jstar, istar] = -sin(theta)
    G[jstar, jstar] = cos(theta)
    KAC = KAC %*% G
  }
  KAC = KAC * sqrt(vlen)

  MSE = tibble(IID = rep(0, length(n.feature)),
               CIRC = rep(0, length(n.feature)),
               GORT = rep(0, length(n.feature)),
               HD = rep(0, length(n.feature)),
               KAC = rep(0, length(n.feature)))
  for (j in 1:n.pair) {
    # randomly choose two vectors and apply some mechanism to make sure |xy| > 1
    v1 = data.train[sample(c(1:nrow(data.train)), 1), ]
    v2 = data.test[sample(c(1:nrow(data.test)), 1), ]
    if ((v1 %*% v2 >= 0) & (v1 %*% v2 < 1)) v2[14] = 1
    else if ((v1 %*% v2 < 0) & (v1 %*% v2 > -1)) v2[14] = -1

    for (k in 1:length(n.feature)) {

      MSE$IID[k] = MSE$IID[k] + ran_dot(k, IID, v1, v2)
      MSE$CIRC[k] = MSE$CIRC[k] + ran_dot(k, CIRC, v1, v2)
      MSE$GORT[k] = MSE$GORT[k] + ran_dot(k, GORT, v1, v2)
      MSE$HD[k] = MSE$HD[k] + ran_dot(k, HD, v1, v2)
      MSE$KAC[k] = MSE$KAC[k] + ran_dot(k, KAC, v1, v2)
    }
  }
  res = res + MSE / n.pair
}
res = res / n.simu

res %>%
  mutate(features = n.feature) %>%
  gather(key = method, value = MSE, IID:KAC) %>%
  ggplot(aes(x = features, y = MSE, group = method, col = method)) +
  geom_line()
```



Conclusions

Comparing these 4 methods against the unstructured Gaussian matrices (IID, blue line), we can see that

- when number of features is small, Gaussian orthogonal matrices (GORT) are the best
- when number of features is large, random rotation matrices (KAC) and random Hadamard matrices with three HD blocks (HD) performs the best

Problem 2

Theory

For nonisotropic Gaussian kernels:

$$K(x,y) = exp\{-\frac{\tau^T Q \tau}{2}\} = \int_{\mathbb{R}^d} p(\omega) cos(\omega^T \tau) d\omega$$
$$p(\omega) = \frac{1}{\sqrt{2\pi} \cdot det(Q)} exp\{-\frac{\omega^T Q^{-1} \omega}{2}\}$$

where $Q = VV^T, V \in \mathbb{R}^{d \times N}$.

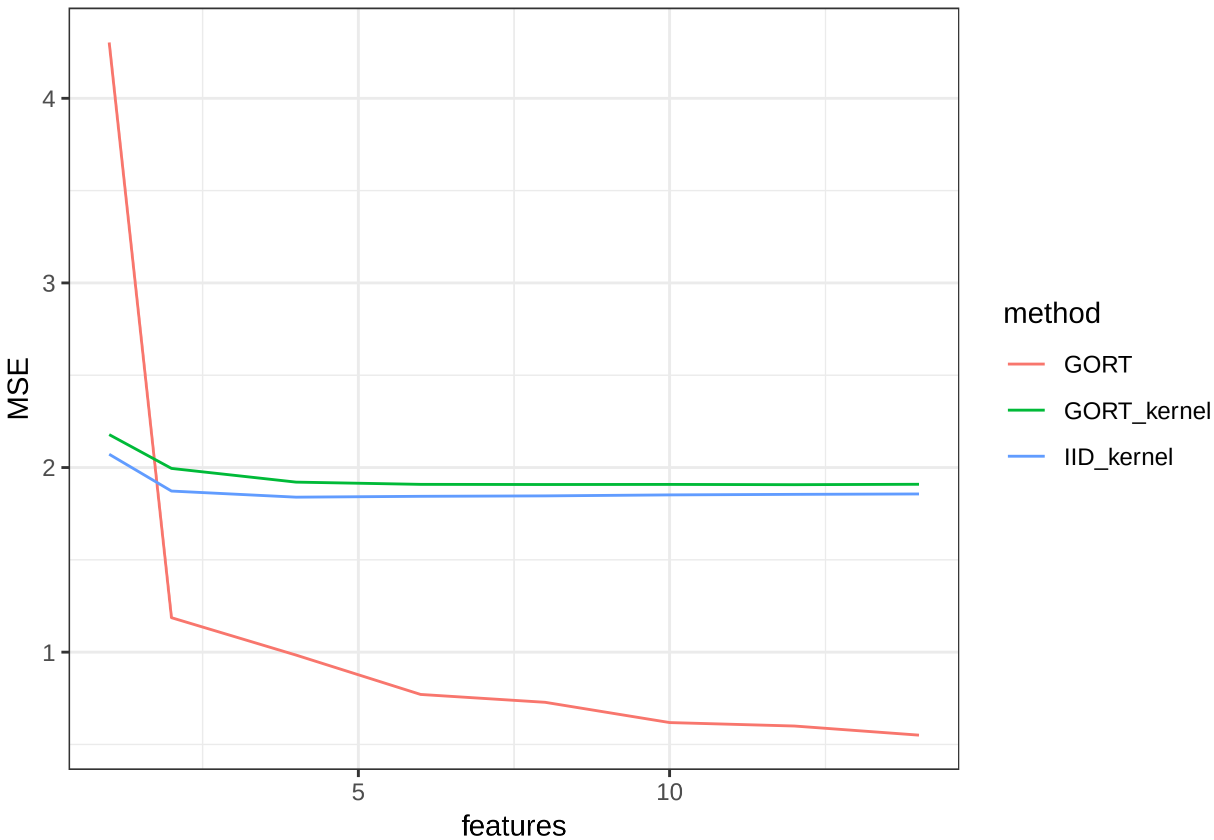
Simulation

$$
\begin{aligned}
\hat{K}(x,y) &= \frac{1}{m}\sum_{i=1}^{m} cos(\omega_i^T(x-y)) \\
&= \frac{1}{m}\sum_{i=1}^{m} cos(\omega_i^T x - \omega_i^T y)) \\
&= \frac{1}{m}\sum_{i=1}^{m} (cos\omega_i^T x \cdot cos\omega_i^T y + sin\omega_i^T x \cdot sin\omega_i^T y) \\
&= <\phi(x), \phi(y)>
\end{aligned}
$$

$$
\phi(x) = \frac{1}{\sqrt{m}}\begin{pmatrix} cos(Gx) \\ sin(Gx) \end{pmatrix}, G = \begin{pmatrix} \omega_1^T \\ \omega_2^T \\ \cdot \\ \cdot \\ \omega_m^T \end{pmatrix}, where <\omega_i, \omega_j>= 0, \forall i \neq j
$$

```
res_K = tibble(GORT = rep(0, length(n.feature)),
               IID_kernel = rep(0, length(n.feature)),
               GORT_kernel = rep(0, length(n.feature)))
ran_K_dot <- function(k, Q, v1, v2) {
  G = Q[sample(c(1:vlen), k, replace = F),] # G with randomly generated m features
  return((v1 %*% v2 - t(rbind(cos(G %*% t(t(v1))), sin(G %*% t(t(v1))))) %*% rbind(cos(G %*% t(t(v2))),
  }

for (i in 1:n.simu) {
  # construct Q
  V = matrix(rnorm(vlen * sample(100, 1)), nrow = vlen)
  Q = V %*% t(V)

  # Gaussian orthogonal matrices
  IID = matrix(rnorm(vlen*vlen), nrow = vlen, ncol = vlen)
  GORT = IID
  for (m in 2:nrow(GORT)) {
    tmp = GORT[m,]
    for (n in 1:(m - 1)) {
      GORT[m,] = GORT[m,] - project(tmp ~ GORT[n,], coefficients = F)
    }
  }

  MSE_K = tibble(GORT = rep(0, length(n.feature)),
                 IID_kernel = rep(0, length(n.feature)),
                 GORT_kernel = rep(0, length(n.feature)))
  for (j in 1:n.pair) {
    # randomly choose two vectors and apply some mechanism to make sure |xy| > 1
    v1 = data.train[sample(c(1:nrow(data.train)), 1), ]
    v2 = data.test[sample(c(1:nrow(data.test)), 1), ]
    if ((v1 %*% v2 >= 0) & (v1 %*% v2 < 1)) v2[14] = 1
    else if ((v1 %*% v2 < 0) & (v1 %*% v2 > -1)) v2[14] = -1

    for (k in 1:length(n.feature)) {
      MSE_K$GORT[k] = MSE$GORT[k] + ran_dot(k, GORT, v1, v2)
      MSE_K$IID_kernel[k] = MSE_K$IID_kernel[k] + ran_K_dot(k, IID, v1, v2)
```

```
      MSE_K$GORT_kernel[k] = MSE_K$GORT_kernel[k] + ran_K_dot(k, GORT, v1, v2)
    }
  }
  res_K = res_K + MSE_K / n.pair
}
res_K = res_K / n.simu

res_K %>%
  mutate(features = n.feature) %>%
  gather(key = method, value = MSE, GORT:GORT_kernel) %>%
  ggplot(aes(x = features, y = MSE, group = method, col = method)) +
  geom_line()
```



Conclusion

By comparing unstructured Gaussian kernels (IID_kernel), Gaussian orthogonal kernel (GORT_kernel) and Gaussian orthogonal matrices (GORT), we can see that:

- the estimation of IID_kernel and GORT_kernel are steadier thanGaussian orthogonal matrices
- when number of features is extremely small, kernel-based methods are better
- when number of features is large, Gaussian orthogonal matrices are better

Time complexity

- The time complexity for unstructured Gaussian kernels is $O(d) = d \cdot logd$

- The time complexity for Gaussian orthogonal kernels is $O(d) = d^3$, because it includes Gram-Schmidt orthogonalization
- The time complexity for GORT is $O(d) = d^3$, because it includes Gram-Schmidt orthogonalization