Principal coordinate ridge regression: A toy example

Philip T. Reiss, David L. Miller, Pei-Shien Wu and Wen-Yu Hua

2016-06-16

Generating the toy data

The accompanying paper (Reiss et al. 2016) introduces a toy example in which scalar-on-function linear regression fails but the proposed method, principal coordinate ridge regression (PCoRR), works well with a judiciously chosen distance. Here each of the functional predictors $x_i(t)$, $i = 1, \ldots, 30$, is a noisy version of a function on $[0, 1]$ that equals zero except around $t = \tau_i$, with a negative blip just before and a positive blip just after that point.

We generate the functional predictors in three steps:

1. Create a 30×101 zero matrix `Xnl` representing the 30 functions sampled at $0, .01, \ldots, 1$.

```
Xnl <- matrix(0,30,101)
set.seed(813)
tau <- sort(runif(30, .04, .96))
```

2. For each function, randomly choose the location τ_i for the negative and positive blips.

```
for (i in 1:30) {
    last.neg <- ceiling(100*tau[i])
    first.pos <- last.neg+1
    Xnl[i, (last.neg-4):(last.neg)] <- -1
    Xnl[i, (first.pos):(first.pos+4)] <- 1
}
```

3. Add white noise to `Xnl` to get the final functional predictor matrix `X.toy`.

```
X.toy <- Xnl + matrix(rnorm(30*101, sd=.05), 30)
```

For each i, the response y_i equals τ_i plus mean-zero noise:

```
y.toy <- tau + rnorm(30, sd=.01)
```

Figure 1 shows two functional predictors without noise (left panel) along with their corresponding observed functions (middle), and a color-coded plot of 30 functional predictors (right). Plots were generated by the following code:

```
par(mfrow=c(1,3))

matplot((0:100)/100, t(Xnl[c(4,25), ]), type="l", xlab="t", ylab="",
        ylim=range(X.toy), main="Noiseless functions")

matplot((0:100)/100, t(X.toy[c(4,25), ]), type="l", xlab="t", ylab="",
        ylim=range(X.toy), main="Observed functions")
```

```
matplot((0:100)/100, t(X.toy), type="l", lty=1, xlab="t", ylab="",
        col=rainbow(30, end=0.9)[(y.toy-min(y.toy))/diff(range(y.toy))*29+1],
        main="Rainbow plot")
```

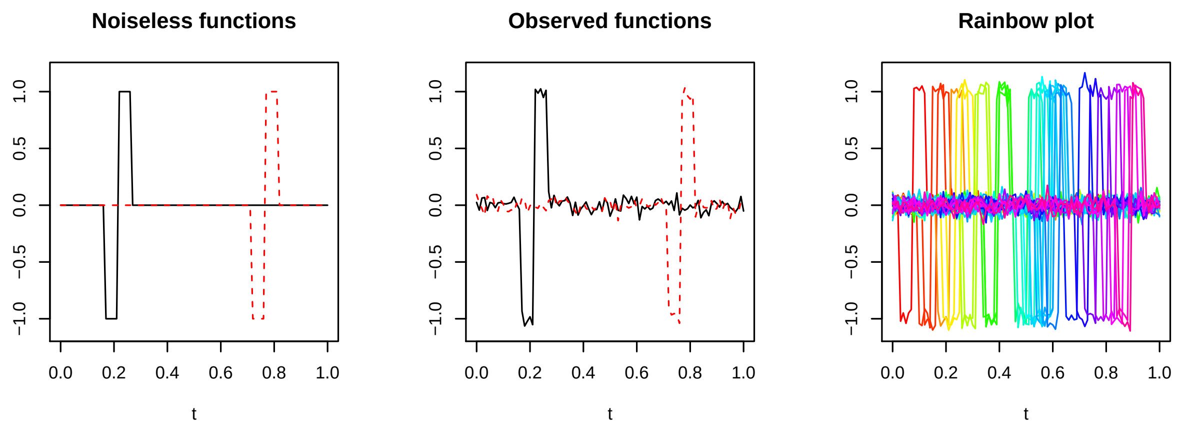

Figure 1: Two instances of the noiseless functional predictors (left panel), the corresponding observed functions x_i (middle panel), and all 30 functional predictors, color-coded by the corresponding responses y_i (right panel).

Dynamic time warping distances among the functional predictors

We obtain windowed dynamic time warping (DTW) distances among the functions as follows:

```
require(dtw)
D.dtw <- dist(X.toy, method="dtw", window.type="sakoechiba", window.size=5)
```

Figure 2 displays the windowed DTW distance matrix D_{dtw} among the 30 functions, arranged in order of τ_i. Two functions have low distance if and only if they are "neighbors" (have similar τ_i). The other subfigures show how the distances in q-dimensional principal coordinate space approach D_{dtw} as q increases.

Figure 2 is generated by the following code:

```
cmd <- cmdscale(D.dtw, eig = TRUE, k = 29)
npco <- c(1, 3, 5, 9)
Dl <- array(NA, c(30, 30, 5))

for (k in 1:4) {
    Dl[, , k] <- as.matrix(dist(cmd$points[, 1:npco[k]]))
}

Dl[, , 5] <- as.matrix(D.dtw)
par(mar = c(5, 0, 1, 2) + 0.1, oma = c(0, 0, 0, 1), mfrow = c(1, 5))

require(fields)
for (k in 1:5) {
    image(Dl[, 30:1, k], main = ifelse(k < 5, paste(npco[k], "PCos"), "Exact distance"),
        zlim = range(Dl), axes = FALSE, col = tim.colors(), asp = 1)
}

par(oma = c(0, 0, 0, 0.5))
image.plot(legend.only = TRUE, zlim = range(Dl), legend.width = 2.5, legend.mar = 6,
    asp = 1)
```

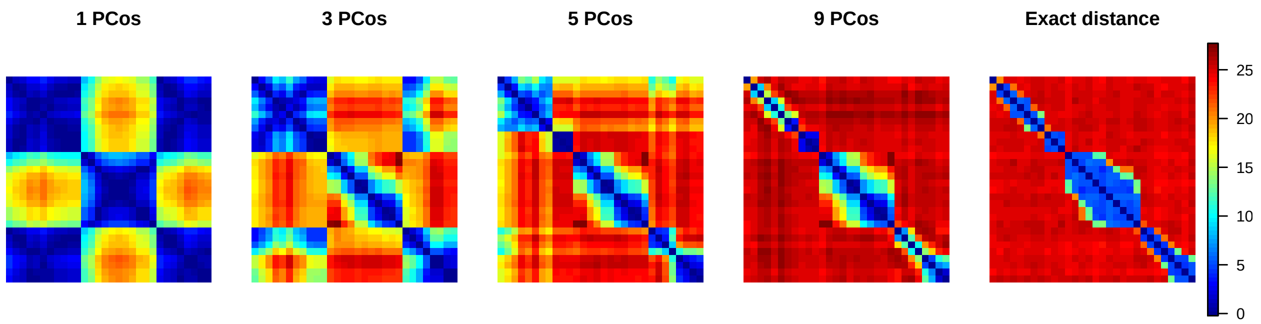

Figure 2: Windowed DTW distance matrix among the 30 functions, in order of $ au_i$. Left to right show how the distances in q-dimensional principal coordinate space approach those in the distance matrix as the number of PCos increases.

Note that some of the `k=29` eigenvalues obtained above will not be > 0 and this will, of course differ depending on the data set being analysed. In practice we generally do not worry about this as the distance matrix is approximated will by some number of dimensions $\ll k$.

Comparing PC vs. PCo ridge regression

We use generalized cross-validation (GCV) as a performance measure to compare ridge regression on *principal components* (a linear model) vs. on DTW-based *principal coordinates* (a nonparametric model). First we set up a matrix for the GCV values and load the required packages. As of this writing, the proposed method is implemented in the `poridge` package, available at https://github.com/dill/poridge; we plan to incorporate that package within the next release of the `refund` package on CRAN.

```
GCVmat <- matrix(NA, 15, 2)
require(mgcv)
```

```
## Warning: package 'nlme' was built under R version 3.2.4
```

```
require(poridge)
dummy <- rep(1, 30)
```

The pseudo-response vector `dummy` is needed for the `gam` syntax below. We then loop through each candidate number `k.` of PCos from 1 to 15. For each `k.` we fit two models by the `mgcv::gam` basis type `"pco"` implemented by `poridge`. For the first fit (`m1`), the PCos are based on Euclidean distance `dist` among the rows of the raw functional predictor data `X.toy`; these PCos are equivalent to principal components. For the second fit (`m2`), the PCos are based on the windowed DTW distances among the functional predictors. This code fragment illustrates the two ways to input the distance information: one can supply either a data matrix `realdata` and distance function `dist_fn` as in `m1`, or a distance matrix `D` as in `m2`.

```
for (k. in 1:15) {
    m1 <- gam(y.toy ~ s(dummy, bs="pco", k=k.,
                        xt=list(realdata=X.toy, dist_fn=dist, fastcmd=FALSE)),
              method="REML")
    m2 <- gam(y.toy ~ s(dummy, bs="pco", k=k.,
                        xt=list(D=D.dtw, fastcmd=FALSE)),
              method="REML")
    GCVmat[k., ] <- 30 * c(sum(m1$residuals^2) / m1$df.residual^2,
                           sum(m2$residuals^2) / m2$df.residual^2)
}
```

As we can see in the middle panel of the Figure 3, PCoRR based on DTW distance attains much lower GCV values than ridge regression on leading PC's.

```
par(mfrow=c(1,3))
cexc <- 10

image.plot(as.matrix(D.dtw)[ , 30:1], axes=FALSE,
           main="DTW distances among observations", asp=1,
           legend.cex=cexc, legend.width=2.5)

par(mar=c(5.1, 4.9, 4.1, 1.6))

matplot(GCVmat, lty=1:2, col=1, pch=16:17, type="o", ylab="GCV",
        xlab="Number of principal components / coordinates",
        main="Predictive performance")
legend("right", c("PCR", "DTW-based PCoRR"), lty=1:2, pch=16:17)

# Hat matrix for fit with 10 PCos
gg <- gam(y.toy ~ s(dummy, bs="pco", k=10, xt=list(D=D.dtw, fastcmd=FALSE)),
          method="REML")

hatmat <- cmd$points[,1:10] %*% diag(1/(cmd$eig[1:10]+gg$sp)) %*%
           t(cmd$points[,1:10]) + matrix(1/30,30,30)

image.plot(hatmat[,30:1], axes=FALSE, main="Hat matrix for PCoRR",
           legend.cex=cexc, legend.mar=6, legend.width=2.5, asp=1)
```

The hat matrix for DTW-based PCoRR with 10 PCos, shown in the right panel of Figure 3, provides some insight into why this method works well. This hat matrix is an approximate color-inversion of the DTW distance matrix (Figure 3, left). Thus observations with lower distance to the ith observation tend to have higher weight in the regression.

Bibliography

Reiss, Philip T, David L Miller, Pei-Shien Wu, and Wen-Yu Hua. 2016. "Penalized Nonparametric Scalar-on-Function Regression via Principal Coordinates."

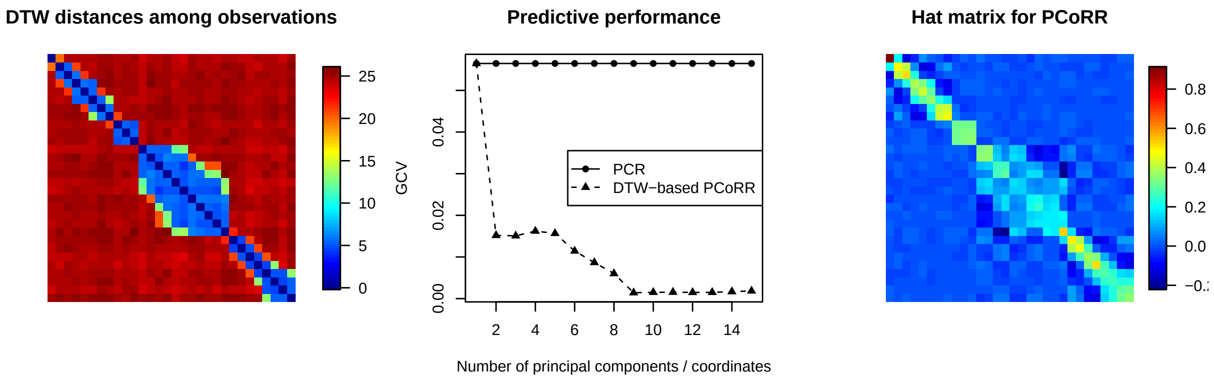

Figure 3: Left: DTW distance matrix; middle: GCV performance of PCR and DTW-based PCoRR methods; right: hat matrix for the DTW-based PCoRR model with 10 PCos.