Journal of Statistical Software

April 2006, Volume 15, Issue 10. http://www.jstatsoft.org/

CVThresh: R Package for Level-Dependent Cross-Validation Thresholding

Donghoh Kim
Hongik University

Hee-Seok Oh
Seoul National University

Abstract

The core of the wavelet approach to nonparametric regression is thresholding of wavelet coefficients. This paper reviews a cross-validation method for the selection of the thresholding value in wavelet shrinkage of Oh, Kim, and Lee (2006), and introduces the R package **CVThresh** implementing details of the calculations for the procedures.

This procedure is implemented by coupling a conventional cross-validation with a fast imputation method, so that it overcomes a limitation of data length, a power of 2. It can be easily applied to the classical leave-one-out cross-validation and K-fold cross-validation. Since the procedure is computationally fast, a level-dependent cross-validation can be developed for wavelet shrinkage of data with various sparseness according to levels.

Keywords: h-likelihood, imputation, missing values, R, shrinkage, thresholding.

1. Introduction

The main goal of this paper is to introduce the package **CVThresh** in R, which implements level-dependent cross-validation for thresholding in wavelet shrinkage by Oh, Kim, and Lee (2006). In addition, we provide algorithmic calculations and details of the procedure which could not be given in the original paper, so one can be ready for applying **CVThresh** to real data. The **CVThresh** package is implemented in R. The source code, windows binary and reference manual are available from `http://stats.snu.ac.kr/~heeseok/cv.html` and the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/` (Kim and Oh, 2006). See reference manual for usage and syntax of **CVThresh** for implementing level-dependent cross-validation for wavelet shrinkage. For a step-by-step tutorial one may download example R codes and other introductory documentation from `http://stats.snu.ac.kr/~heeseok/cv.html`. Note that **CVThresh** package is implemented with **WaveThresh3** or newer and **EBayesThresh** which are available from `http://www.stats.bris.ac.uk/~wavethresh/` (Nason, 1998) and CRAN (Silverman, 2004), respectively. For

detailed explanation of methodology for **EBayesThresh** package, see Johnstone and Silverman (2005).

The statistical problem we attempt to solve with wavelet shrinkage is, given the data y_i's observed from the model $y_i = f(x_i) + \epsilon_i$ $(i = 1, 2, \ldots, n = 2^J)$, to estimate f. The standard wavelet shrinkage can be performed in the following steps: (1) taking the discrete wavelet transform of y_i; (2) processing the resulting coefficients by some procedure; and (3) transforming back to obtain the estimate $\hat{f}$. The part (2) to determine a threshold value plays the most important role for the estimation quality. Oh, Kim, and Lee (2006) proposed a level-dependent cross-validation for selection of thresholding value(s).

2. Cross-validation scheme

When establishing a statistical model based on the observations at hand, it is hard to evaluate how well a statistical model predicts new observations. One way to overcome this problem is cross-validation. The procedure of cross-validation consists of following steps

1. **CV Scheme step:** deciding a scheme how to choose test dataset, which is regarded as new observations, and
2. **Optimization step:** estimating a statistical model based on the remaining data called training dataset, and evaluating models by prediction error of test dataset.

The first step is to decide test datasets, which can be constructed through re-indexing of observation index. That is, with defining a function $k : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, K\}$, we obtain K test datasets, $T_j = \{y_i : j = k(i),\ i = 1, 2, \ldots, n\}$ $(j = 1, \ldots, K)$. According to the scheme of re-index, we can implement various kinds of K-fold cross-validations. Thus, one constructs K test datasets in a way that each test dataset consists of blocks of b consecutive data. In addition, one can employ a randomized scheme to select a block or a fold. Such a cross-validation scheme can be referred as a random K-fold cross-validation with block size b. The re-indexing scheme characterizes various cross-validation schemes.

The R function `cvtype()` provides indexes of test data according to various cross-validation schemes. In other words, `cvtype()` generates an index set of each test data of K-fold. For example, for performing a random 4-fold cross-validation with block size 2 of observations $y_1, \cdots, y_{32}$ from Heavisine function (Donoho and Johnstone, 1994), the following R code produces an index set of each test dataset.

```
> set.seed(3)
> cvtype(n=32, cv.bsize=2, cv.kfold=4, cv.random=T)$cv.index
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    9   10   11   12   15   16   25   26
[2,]    4    5   10   11   16   17   26   27
[3,]   16   17   22   23   28   29   30   31
[4,]    4    5    8    9   20   21   24   25
```

Each row represents an index set corresponding to test dataset T_j $(j = 1, 2, 3, 4)$. That is, each training set consists of data excluding T_j in turn. See the left panel of Figure 1.

For the traditional K-fold cross-validation in the right panel of Figure 1, use the following R code.

```
> cvtype(n=32, cv.bsize=1, cv.kfold=4, cv.random=F)$cv.index
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    5    9   13   17   21   25   29
[2,]    2    6   10   14   18   22   26   30
[3,]    3    7   11   15   19   23   27   31
[4,]    4    8   12   16   20   24   28   32
```

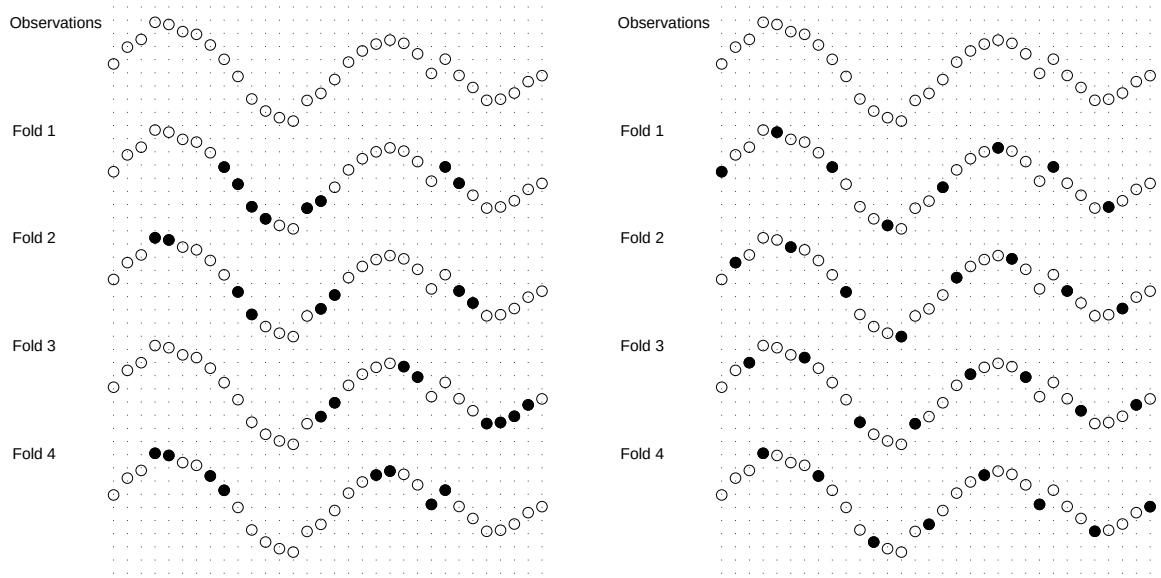

Figure 1: Circles are 32 observations from Heavisine. Black circles are each test dataset of a random and the traditional 4-fold cross-validation, respectively.

If one would like to construct folds by just specifying the number of blocks with a block size, use `cvtype()` in the following manner. In this example code, the number of blocks is 2 with block size 2.

```
> set.seed(1)
> nblock <- 2; cv.bsize <- 2; cv.kfold <- trunc(32/(cv.bsize*nblock))
> cvtype(n=32,cv.bsize=cv.bsize,cv.kfold=cv.kfold,cv.random=T)$cv.index
```

```
     [,1] [,2] [,3] [,4]
[1,]   11   12   17   18
[2,]    8    9   26   27
[3,]   18   19   20   21
[4,]    5    6    7    8
[5,]   12   13   22   23
[6,]   23   24   29   30
```

```
[7,]   25   26   29   30
[8,]    3    4   21   22
```

For constructing test datasets with block size 3 where each test dataset consists of approximately 33% of the whole data, try the following code.

```
> set.seed(1)
> perc <- c(0.33); cv.bsize <- 3; cv.kfold <- trunc(1/perc)
> cvtype(n=32,cv.bsize=cv.bsize,cv.kfold=cv.kfold,cv.random=T)$cv.index
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]   10   11   12   16   17   18   22   23   24
[2,]   16   17   18   25   26   27   28   29   30
[3,]    2    3    4    5    6    7   26   27   28
```

3. Imputation by wavelet and h-likelihood

In wavelet context, the classical cross-validation method cannot be directly applied to wavelet shrinkage, because the limitation of Mallat's fast algorithm for discrete wavelet transform is of the length of data$= 2^J$ where J is an integer. To overcome this problem, we propose the cross-validation which is described as the following three steps:

1. **CV Scheme step:** deciding a scheme how to construct test datasets,
2. **Imputation step:** performing the imputation method for test datasets, and
3. **Optimization step:** finding thresholding values minimizing prediction error.

In Section 2, CV Scheme step using the R function `cvtype()` has already been explained. This section introduces a simple methodology for imputation step based on a hierarchical likelihood (or h-likelihood) concept and the R function `cvimpute.by.wavelet()` for imputation.

Lee and Nelder (1996, 2001) introduced the hierarchical likelihood as an extended likelihood for general models that include unobserved random variables such as missing. Following Lee and Nelder (1996, 2001), we impute the missing values by maximizing the h-likelihood. Since it has been known that a wavelet shrinkage estimator can be formulated by penalized least squares problem (Antoniadis and Fan, 2001), we obtain the wavelet estimate by minimizing the penalized least squares criterion after missing values are imputed. These arguments lead to the following iterative algorithm. See Oh, Kim, and Lee (2006) for details.

Suppose that $y = (y_{obs}, y_{mis})$ are random variables with mean f and variance σ^2, where $y_{obs} = (y_1, \ldots, y_k)$ are $y_{mis} = (y_{k+1}, \ldots, y_n)$ denote the subsets of observed data and missing data, respectively. Then the imputation step can be described as

1. Set an initial estimate $\hat{y}_{mis}^{(0)}$.
2. Iterate at l until converge.

   2-1 Estimate $\hat{f}^{(l)}$ by applying a wavelet shrinkage to the new data $y_{new}^{(l)} = \left(y_{obs}, \hat{y}_{mis}^{(l-1)}\right)$.

2-2 Impute $\hat{y}_{mis}^{(l)}$.

The R function `cvimpute.by.wavelet()` imputes test data by the proposed methodology, given the index of test dataset provided by a cross-validation scheme. Suppose that we have 1024 observations from the model $y_i = f_i + \epsilon_i$, where f is Heavisine and noise ϵ's are iid Gaussian random variables. Figure 2 shows imputation results at iterations 1, 2, and 7, when randomly chosen 64 observations are treated as missing values. As seen, the method converges very fast.

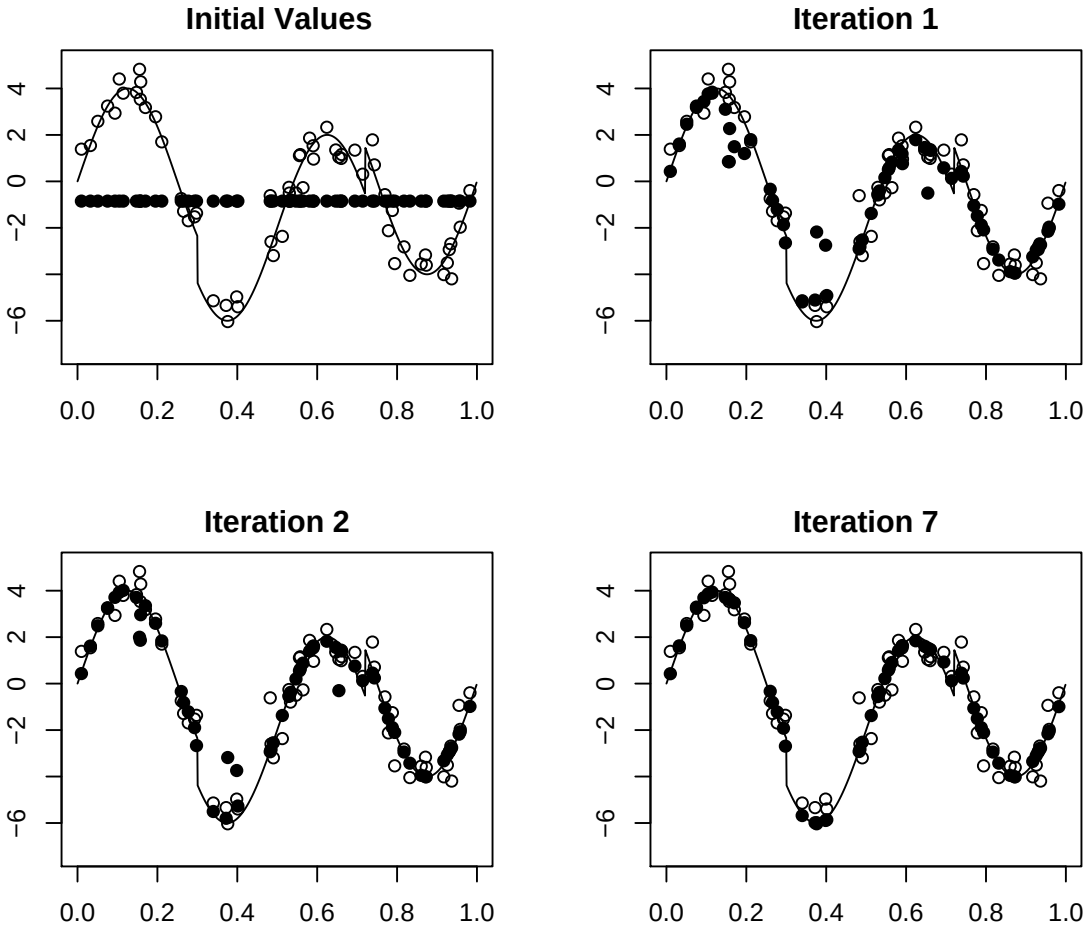

Figure 2: Black circles are imputed values at iteration 1, 2 and 7 with mean values of y as the initial values, and circles are original observations.

The following code implements the imputation of each test data for a random 8-fold cross-validation with block size 2. Figure 3 represents a "goodness of fit" of imputation results for the 8-fold cross-validation. Note that throughout the imputation process, **EBayesThresh** has been used for threshold value(s).

```
> ### Define CV scheme
> set.seed(1)
```

```
> cv.index <- cvtype(n=1024, cv.bsize=2, cv.kfold=8, cv.random=T)$cv.index
> ### Generate observations
> snr <- 5
> testdata <- heav(1024)
> x <- testdata$x
> y <- testdata$meanf + rnorm(1024, 0, testdata$sdf/snr)
> ### Run imputation
> yimpute <- cvimpute.by.wavelet(y=y, impute.index=cv.index)$yimpute
> ### Plot of imputation results
> par(mar=0.1+c(4,4,2,1))
> plot(y, yimpute, xlab="Observations", ylab="Imputed Values",
       main="Piecewise Polynomial", cex=0.5);abline(0,1)
```

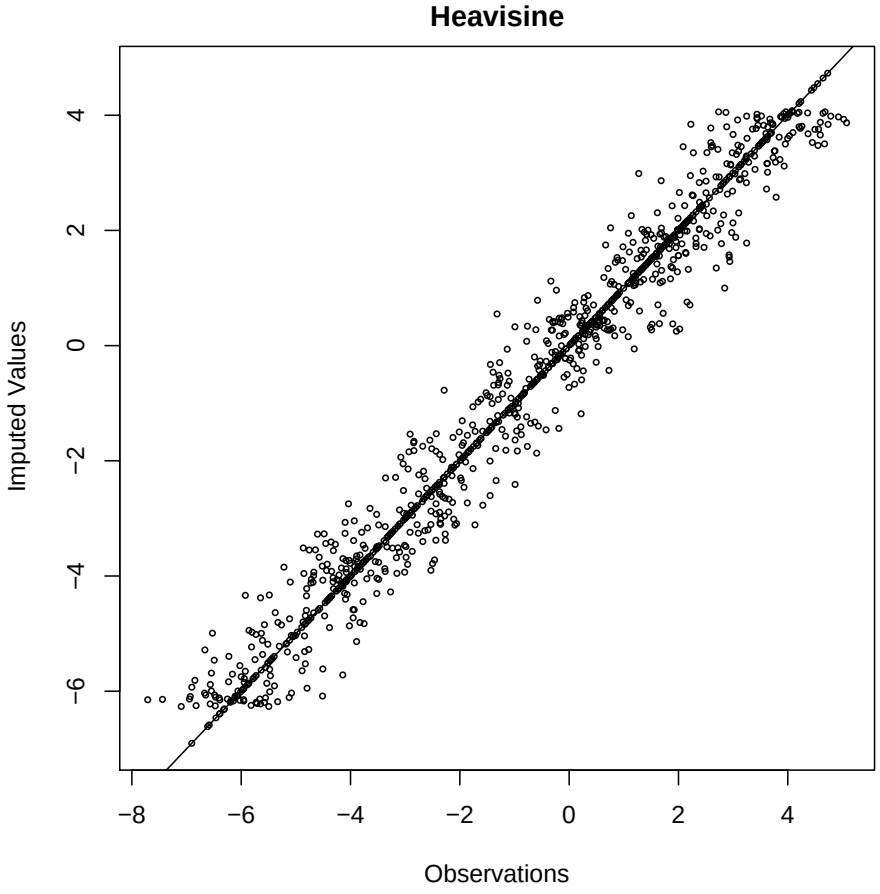

Figure 3: A goodness of fit of imputation results obtained in Figure 2.

4. Optimization after imputation

Consider the classical K-fold cross-validation in wavelet context and prediction error CV. The CV score can be calculated by expelling the kth part of data. That is, the CV score of a

K-fold cross-validation with indexing $k : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, K\}$ can be defined as

$$CV(\lambda) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}_{\lambda}^{-k(i)}(x_i) \right)^2, \tag{1}$$

where $\hat{f}_{\lambda}^{-k(i)}(x_i)$ denotes the wavelet estimate at x_i given a threshold λ with removing the kth part of data.

After applying the imputation procedure, we have a new dataset $y_{new} = (y_{obs}, \hat{y}_{mis})$. We are now free to use Mallat's fast algorithm. Finally we find a value $\hat{\lambda}$ to minimize (1), that is

$$\hat{\lambda} = \arg\min_{\lambda} CV(\lambda).$$

The R function `cvwavelet.after.impute()` calculates the optimal threshold values, and then reconstruct f from noisy data y. The inputs of the R function `cvwavelet.after.impute()` are listed as the index of each test data, imputed values according to cross-validation scheme, and discrete wavelet transform of y. For an efficient computation of $CV(\lambda)$, we adapt the grid search algorithm of the R function `WaveletCV()` in **WaveThresh3** of Nason (1998).

Although one can use any cross-validation scheme and imputation methods, it is convenient to use the R functions `cvtype()` and `cvimpute.by.wavelet()` for those processes. The following example shows how to conduct a wavelet shrinkage with the threshold(s) by the proposed method for a real data `IPD`. The dataset `IPD` used in Nason (1996) contains 4096 observations of inductance plethysmography data regularly sampled at 50Hz starting at 1229.98 seconds. We estimates the mean function by a random 4-fold cross-validation with block size 2.

```
> ### Define IPD
> data(ipd)
> y <- as.numeric(ipd); n <- length(y); nlevel <- log2(n)
> ### Define CV scheme
> set.seed(1)
> cv.index <- cvtype(n=n, cv.bsize=2, cv.kfold=4, cv.random=T)$cv.index
> ### Run imputation
> yimpute <- cvimpute.by.wavelet(y=y, impute.index=cv.index)$yimpute
> ### Discrete wavelet transform of observation
> ywd <- wd(y)
> ### Find optimal threshold values and reconstruct the data
> out <- cvwavelet.after.impute(y=y, ywd=ywd, yimpute=yimpute,
                    cv.index=cv.index, cv.optlevel=c(3:(nlevel-1)))
```

Cross-validation can be performed as a level-by-level-wise by applying K-fold cross-validation algorithm to each level or groups of resolution levels from coarse level, say 3 to the finest level, say l. This level-dependent cross-validation finds thresholding values $\left(\hat{\lambda}_1, \ldots, \hat{\lambda}_l\right)$ to minimize prediction error CV, that is

$$\left(\hat{\lambda}_1, \ldots, \hat{\lambda}_l\right) = \arg\min_{\lambda_1, \ldots, \lambda_l} CV(\lambda_1, \ldots, \lambda_l),$$

where

$$CV(\lambda_1, \ldots, \lambda_l) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}_{\lambda_1, \ldots, \lambda_l}^{-k(i)}(x_i) \right)^2.$$

We need to specify which levels or groups of levels are thresholded. The argument `cv.optlevel` of the R function `cvwavelet.after.impute()` specifies thresholding structure. See the following R codes which define thresholding structures.

```
> ### Threshold (level 3 to the finest level) at the same time.
> cv.optlevel <- 3

> ### Threshold two groups of resolution levels, (level 3, 4) and
> ### (level 5 to the finest level).
> cv.optlevel <- c(3, 5)

> ### Threshold each resolution level 3, 4, 5, 6, 7, and 8.
> cv.optlevel <- c(3,4,5,6,7,8)
```

The R function `cvwavelet()` implements all three steps for cross-validation scheme, imputation, and thresholding process. In fact, the `cvwavelet()` consists of three R functions `cvtype()`, `cvimpute.by.wavelet()` and `cvwavelet.after.impute()`. A comparison using `IPD` data is performed by **EBayesThresh**, Nason's cross-validation and level-dependent cross-validation below. See Figure 4 for the results indicating that proposed method produces the most smooth reconstruction around 1300 seconds while it keeps the sharpness of observations.

```
> ### level-dependent cross-validation
> set.seed(1)
> out <- cvwavelet(y=y, ywd=ywd, cv.optlevel=c(3:(nlevel-1)),
                                              cv.bsize=2, cv.kfold=4)
> ### Nason's cross-validation
> yn <- wr(threshold(ywd, policy="cv"))
> ### EBayesThresh
> ye <- wr(ebayesthresh.wavelet(ywd, smooth.levels=nlevel-3))
> ### Plotting the results
> par(mfrow=c(2,2), oma=c(0,0,0,0), mar=0.1+c(4,2,2.5,1))
> ts.plot(ts(y, start=1229.98, deltat=0.02, frequency=50),
        main="Observations", xlab = "Seconds", ylab="")
> ts.plot(ts(ye, start=1229.98, deltat=0.02, frequency=50),
        main="EBayes", xlab = "Seconds", ylab="")
> ts.plot(ts(yn, start=1229.98, deltat=0.02, frequency=50),
        main="cv.Nason", xlab = "Seconds", ylab="")
> ts.plot(ts(out$yc, start=1229.98, deltat=0.02, frequency=50),
        main="cv.level", xlab = "Seconds", ylab="")
```

5. Application to image reconstruction

The proposed method is straightforward to extend to a cross-validation method for image data. Similarly, we divide two-dimensional data into K roughly equal-sized parts, and then follow the same steps for one dimensional signal described above. Likewise one-dimensional case, we provide four R functions for images

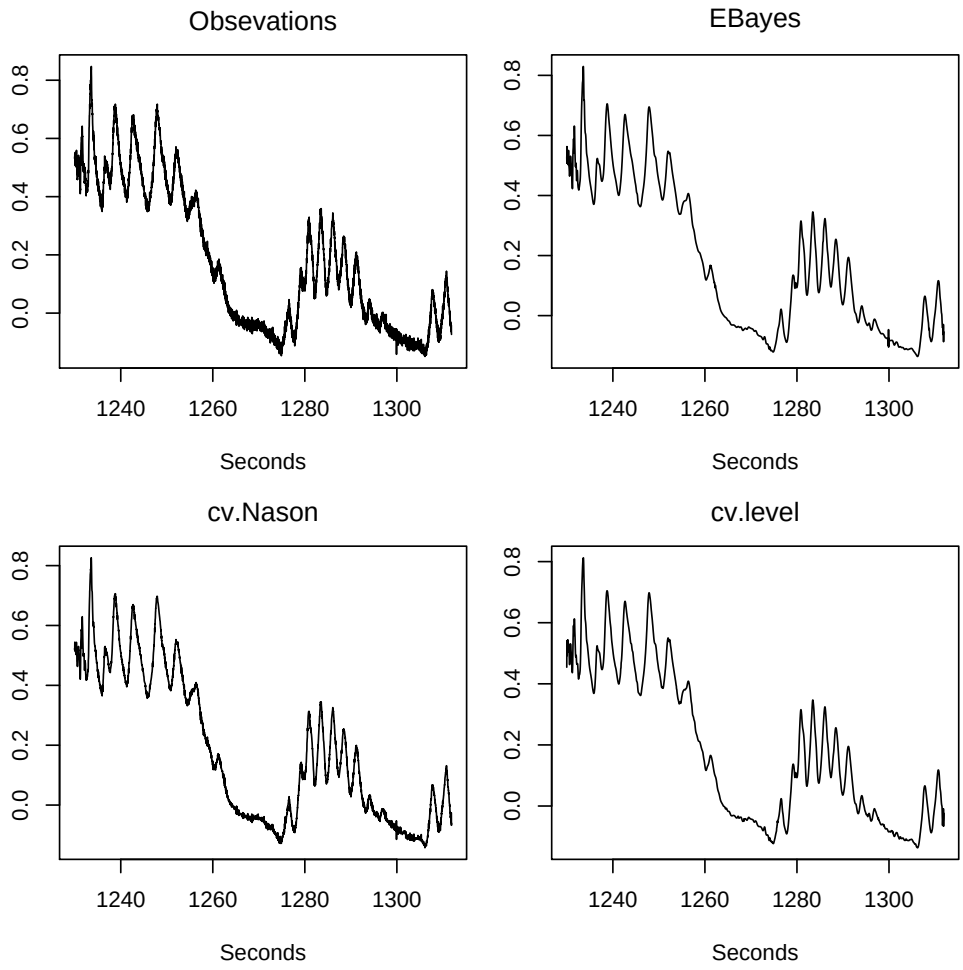

Figure 4: Wavelet estimates of `IPD` data by **EBayesThresh**, Nason's CV, and level-dependent cross-validation performed by a random 4-fold with block size 2.

(1) `cvtype.image()` for re-indexing of two-dimensional cross-validation scheme,

(2) `cvimpute.image.by.wavelet()` for imputation given cross-validation scheme,

(3) `cvwavelet.iamge.after.impute()` for thresholding values and a reconstruction given a cross-validation scheme and imputation, and

(4) `cvwavelet.image()` for combining the above three R functions.

For two-dimensional cross-validation scheme, one needs to specify size of images, two dimensional block size, and the number of folds. Only random scheme is possibly provided.

```
> cvtype.image(n=c(256,256), cv.bsize=c(2,2), cv.kfold=10)
```

The usage of the `cv.optlevel` in the R functions `cvwavelet.iamge.after.impute()` and `cvwavelet.image()` is different from that of one dimensional case. Each level from the lowest resolution to the finest resolution level specified in the `cv.optlevel` is thresholded, and LH,

HL and HH indicating horizontal, vertical, and diagonal detail coefficients are respectively thresholded within a level. One may alter the R code to use a specific thresholding scheme.

To check the performance of the proposed method, noise is added to the original image `lennon`: Gaussian mixture $0.9\phi(\cdot;\text{snr}{=}5)+0.1\phi(\cdot;\text{snr}{=}0.5)$ in Figure 5. Comparing universal, **EBayesThresh**, and level-dependent 10-fold cross-validation threshold, the level-dependent cross-validation outperforms with respect to MSE as well as to visual perception. Especially, level-dependent cross-validation effectively smooths out the extreme pixels from the Gaussian mixture.

```
> ### Generate Noisy Image
> data(lennon)
> sdimage <- sd(as.numeric(lennon))
> nlennon <- ncol(lennon)
> nlevel <- log2(ncol(lennon))
> optlevel <- c(3:(nlevel-1))

> set.seed(55)
> n1 <- trunc(nlennon^2 * 0.9)
> n2 <- nlennon^2 - n1
> lennonnoise <- lennon + matrix(sample(c(rnorm(n1, 0, sdimage/5),
                                rnorm(n2, 0, sdimage*2))), nlennon, nlennon)

> par(mfrow=c(2,2), oma=c(0,0,0,0), mar=0.1+c(1,1,2.0,1))
> image(lennonnoise, axes=F, col=gray(0:100/100), main="Noisy Image")

> ### Reconstruction by Universal Threshold
> lennonwd <- imwd(lennonnoise)
> lennonuv <- imwr(threshold(lennonwd, policy="universal"))
> image(lennonuv, axes=F, col=gray(0:100/100), main="Universal Threshold")

> ### Reconstruction by EBayesThresh
> tmp <- c(8,9,10)
> slevel <- NULL
> for (j in 1:(diff(range(optlevel))+1))
     slevel <- c(slevel, tmp+4*(j-1))
> slevel <- slevel[length(slevel):1]

> sdev <- mad(c(lennonwd[[8]], lennonwd[[9]], lennonwd[[10]]))

> lennonwdth <- lennonwd
> for (j in 1:length(slevel))
  lennonwdth[[slevel[j]]] <- ebayesthresh(lennonwdth[[slevel[j]]], sdev=sdev)
> lennoneb <- imwr(lennonwdth)
> image(lennoneb, axes=F, col=gray(0:100/100), main="EBayes")

> ### Reconstruction by cv.level
> lennoncv <- cvwavelet.image(images=lennonnoise, imagewd=lennonwd,
```

```
                cv.optlevel=optlevel, cv.bsize=c(1,1), cv.kfold=10)$imagecv
> image(lennoncv, axes=F, col=gray(0:100/100), main="cv.level")

> ### MSE of Universal Threshold, Ebayes and cv.level
> c(mean((lennonuv-lennon)^2), mean((lennoneb-lennon)^2),
                                                  mean((lennoncv-lennon)^2))
```

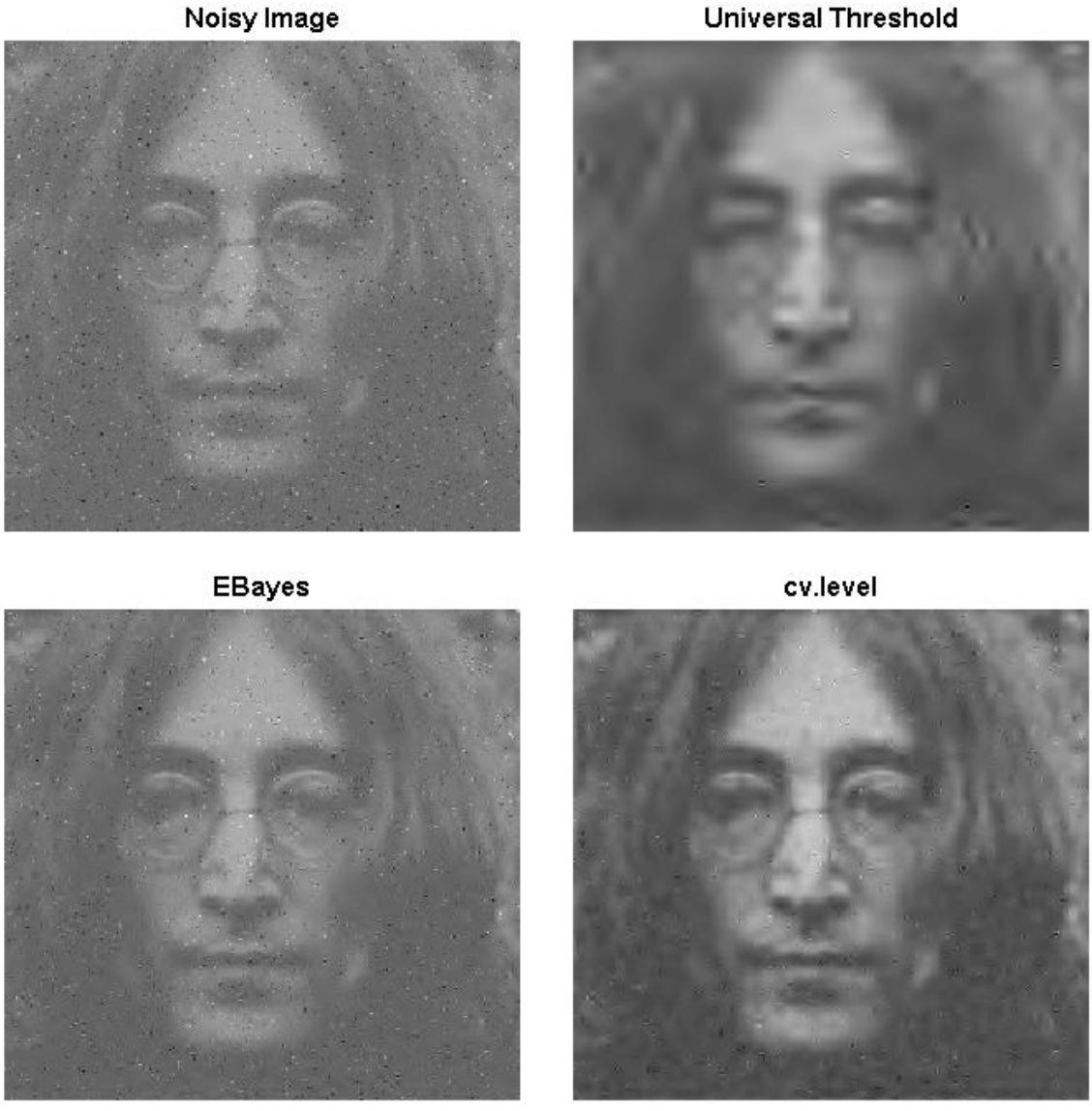

Figure 5: Noisy Lennon image with Gaussian mixture, and wavelet estimates by universal, EBayes, and level-dependent 10-fold CV threshold. MSE's are 250.2, 619.8 and 164.3, respectively.

6. Conclusions

CVThresh is an R package for implementation of the level-dependent cross-validation thresholding in wavelet shrinkage. It can be easily installed by simply loading the package **CVThresh** from the CRAN archive. In this paper, we introduce **CVThresh** and provide a step-by-step tutorial introduction for wide potential applicability. Our hope is that **CVThresh** can help some readers who are interested in the level-dependent cross-validation thresholding to use it for real applications.

Acknowledgements

This research was supported by the Korea Research Foundation Grant funded by the Korean Government(MOEHRD) (KRF-2005-070-C00021).

References

Antoniadis A, Fan J (2001). "Regularization of Wavelet Approximations." *Journal of the American Statistical Association*, **96**, 939–962.

Donoho DL, Johnstone IM (1994). "Ideal Spatial Adaptation by Wavelet Shrinkage." *Biometrika*, **81**, 425–455.

Johnstone IM, Silverman BW (2005). "**EBayesThresh**: R Program for Empirical Bayes Thresholding." *Journal of Statistical Software*, **12**, 1–38.

Kim D, Oh HS (2006). **CVThresh**: Level-Dependent Cross-Validation Thresholding. URL http://CRAN.R-project.org/src/contrib/Descriptions/CVThresh.html.

Lee Y, Nelder JA (1996). "Hierarchical Generalised Linear Models (with Discussion)." *Journal of the Royal Statistical Society B*, **58**, 619–678.

Lee Y, Nelder JA (2001). "Hierarchical Generalised Linear Models: A Synthesis of Generalised Linear Models, Random-Effect Models and Structured Dispersions." *Biometrika*, **88**, 987–1006.

Nason GP (1996). "Wavelet Shrinkage by Cross-Validation." *Journal of the Royal Statistical Society B*, **58**, 463–479.

Nason GP (1998). *WaveThresh3 Software*. Department of Mathematics, University of Bristol, UK. URL http://www.stats.bris.ac.uk/~wavethresh/.

Oh HS, Kim D, Lee Y (2006). "Level-Dependent Cross-Validation Approach for Wavelet Thresholding." *Manuscript for publication*.

Silverman BW (2004). **EbayesThresh**: Empirical Bayes Thresholding and Related Methods. URL http://CRAN.R-project.org/src/contrib/Descriptions/EbayesThresh.html.

Affiliation:

Donghoh Kim
Department of International Management
Hongik University
Jochiwon 339-701, Korea
E-mail: donghohk@hongik.ac.kr

Hee-Seok Oh
Department of Statistics
Seoul National University
Seoul 151-747, Korea
E-mail: heeseok@stats.snu.ac.kr
URL: http://stats.snu.ac.kr/~heeseok/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 15, Issue 10 *Submitted:* 2005-08-22
April 2006 *Accepted:* 2006-04-08