A tutorial on R package dslice

Chao Ye, Bo Jiang, Xuegong Zhang and Jun S. Liu

November 24, 2014

1 Introduction

This `R` package implements dynamic slicing method for dependency detection between a categorical variable and a continuous variable, with applications in non-parametric hypothesis testing, quantitative traits loci (QTLs) study and gene set analysis. Functions to illustrate slicing result and load data for gene set analysis are also provided.

Core functions for testing dependence in `dslice` are implemented in the Cpp language and are integrated in `R` through the `Rcpp` package (Eddelbuettel *et al.*, 2011). Slicing result is illustrated via a `ggplot` object. `dslice` requires package `Rcpp` and `ggplot2`.

2 Running `dslice`

First, we need to install package `Rcpp` and `ggplot2` before we install `dslice`. To load `dslice`, type:

```
> library(dslice)
```

```
Loading required package: Rcpp
Loading required package: ggplot2
Loading required package: scales
```

2.1 A running example of dynamic slicing

To see how dynamic slicing detect the dependency between categorical and continuous variables, let's generate 200 observations of them:

```
> n <- 100
> mu <- 0.5
> y <- c(rnorm(n, -mu, 1), rnorm(n, mu, 1))
> x <- c(rep("G1", n), rep("G2", n))
```

Please note that the type of `x` could be either integer or character. `dslice` provides a function named "relabel" to convert categorical `x` to integers started from 0, which is the standard input of `ds_k` and `ds_eqp_k`:

```
> x <- relabel(x)
```

Following the relabeling step, we can see how many different values the categorical variable have (notice that `x` started with 0, just like the value of array index in C language):

```
> xdim <- max(x) + 1
```

As will be shown in Section 3, to preform dynamic slicing, categorical variables should be sorted according to values of continuous variable:

```
> x <- x[order(y)]
```

With these preparations, we can run dynamic slicing by `ds_k`:

```
> lambda <- 1.0
> dsres <- ds_k(x, xdim, lambda, slice = TRUE)
> dsres

$dsval
[1] 14.78292

$slices
    0  1 total
s1 85 43   128
s2 15 57    72
```

where `lambda` is the penalty for adding one more slice.

Dynamic slicing treats the investigated categorical and continuous variables to be independent if the value of dynamic slicing statistic ("DS-statistic") is 0 (or a very small value, *e.g.*, less

than 1e-6), otherwise dynamic slicing treats there exists dependency between them. For this example, we can see that the value of DS-statistic is larger than 0, which means the `x` and `y` here are not independent.

The value of `lambda` is related to a Type-I error rate, which measures the possibility that dynamic slicing misinforms the two variable are not independence in the actual scenario that them are. Currently there are not close form relationship between the value of slicing penalty and Type-I error rate of dynamic slicing. For how to select a proper `lambda` in slicing, please refer to dataset `ds_type_one_error` in this package. The indicator `slice` indicates whether to report slice strategy with DS-statistic or not. If `slice=TRUE`, then we could see the slicing result in a plot (Figure 1):

```
> colnames(dsres$slices) <- c("G1", "G2", "total")
> slice_show(dsres$slices)
```

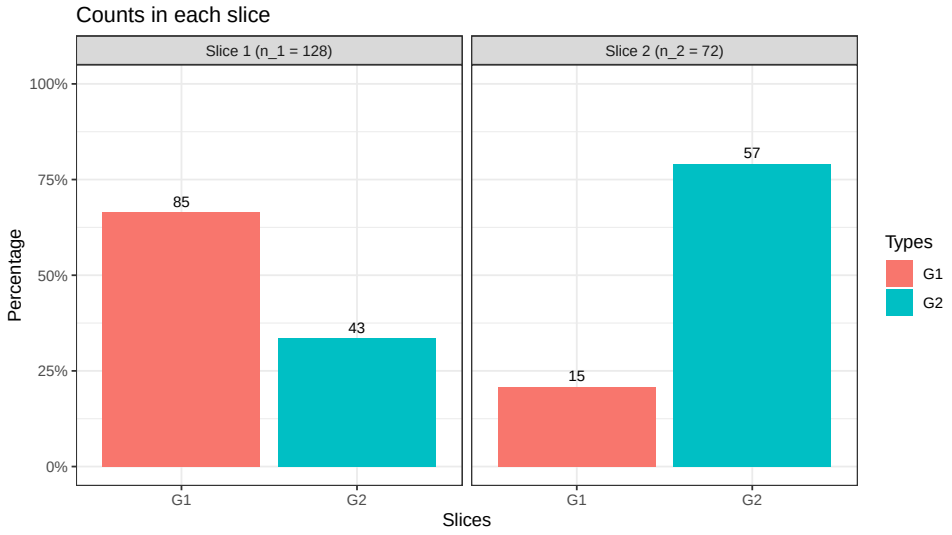

Figure 1: *Illustration of slicing result.*

2.2 A faster version

Jiang *et al.* (2015) showed that the worst case computational complexity of the dynamic slicing algorithm is $O(n^2)$ based on the naive implementation. Though we speed up the computation with additional restriction on splitting clumps of the same `x` values, the computational complexity is still proportional to $O(n^2)$. This order of computational complexity will be embarrassing for problem with large sample size, *e.g.*, all genes of human.

To handle with large number of sample size, we introduce a faster strategy, which is named to dynamic slicing with $O(n^{1/2})$-resolution. The basic idea is almost the same as `ds_k`. The only different is that `ds_eqp_k` groups samples into approximate $O(n^{1/2})$ groups which contain

approximate $O(n^{1/2})$ samples and performs dynamic slicing on their boundaries. The computational complexity is $O(n)$. This much faster version could reduce computation time substantially without too much power loss (Jiang *et al.*, 2015). Based on the strategy of `ds_eqp_k`, we recommend to apply it in large sample size problem and use `ds_k` for ordinary problem. Here is an example to use `ds_eqp_k` (almost the same as `ds_k`):

```
> n <- 100
> mu <- 0.5
> y <- c(rnorm(n, -mu, 1), rnorm(n, mu, 1))
> x <- c(rep("1", n), rep("2", n))
> x <- relabel(x)
> x <- x[order(y)]
> xdim <- max(x) + 1
> lambda <- 1.0
> dsres <- ds_eqp_k(x, xdim, lambda, slice = TRUE)
```

2.3 Dynamic slicing for K-sample ($K \geq 2$) test

One will be aware of the fact that a K-sample ($K \geq 2$) test problem could be viewed as a dependence test of a continuous variable and a categorical variable. Consider the following hypotheses:

$$H_0 : \text{the distributions of } Y \text{ given } X = j \ (1 \leq j \leq K) \text{ are the same}$$
$$\text{v.s. } H_1 : \text{the distributions of } Y \text{ given } X = j \ (1 \leq j \leq K) \text{ are not the same.}$$

Dynamic slicing can be used for testing these hypotheses based on independent observations. `dsclice` provides a function `ds_test` to perform K-sample hypothesis testing:

```
> n <- 100
> y <- c(rnorm(n, -mu, 1), rnorm(n, mu, 1))
> ##  generate x in this way:
> x <- c(rep(0, n), rep(1, n))
> x <- as.integer(x)
> ##  or in this way:
> x <- c(rep("G1", n), rep("G2", n))
> x <- relabel(x)

> lambda <- 1.0
> dsres <- ds_test(y, x, type = "eqp", lambda = 1, rounds = 100)
```

One may find `ds_test` does not require the sorting step of `x` according to `y`. If you do not want to do the rank, this fuction is a good choice. `dslice` still keep functions `ds_k` and `ds_eqp_k`

so that users can also use them if they have a large number of X and do not want to repetitively rank individuals according to Y. For instance, in the eQTL (expression quantitative trait loci) study.

The omnibus function provide `ds_k` (default, `type="ds"`) and `ds_eqp_k` (`type="eqp"`) for K-sample hypothesis testing. Sicne there is not theoretical relationship between `lambda` (penalty of slicing) and Type-I error rate, `ds_test` requires an argument `rounds` to specify the total number of permutations for obtaining an empirical p-value.

2.4 Dynamic slicing for one-sample test

`ds_test` can deal with an one-sample test as well. Assuming that $Y \in \mathbb{R}$ be a univariate continuous variable with unknown cumulative distribution function (CDF) $F(y)$ and probability density function (PDF) $f(y)$. Based on independent observations of Y, $\{y_i\}_{i=1}^n$, we want to test whether the random variable Y follows a distribution with a given CDF $F_0(y)$ and PDF $f_0(y)$. Consider the following hypotheses:

$$H_0 : F(y) = F_0(y) \text{ v.s. } H_1 : F(y) \neq F_0(y).$$

As K-sample test problem, `ds_test` also contains two versions of slicing methods for one-sample test. The usage of `ds_test` is `ds_test(y, x, ..., type = c("ds", "eqp"), lambda = 1, alpha = 1, rounds = 0)`. Examples are here:

```
> ##  One-sample test
> n <- 100
> mu <- 0.5
> y <- rnorm(n, mu, 1)
> lambda <- 1.0
> alpha <- 1.0
> dsres <- ds_test(y, "pnorm", 0, 1, lambda = 1, alpha = 1, rounds = 100)
> dsres <- ds_test(y, "pnorm", 0, 1, type = "ds", lambda = 1, alpha = 1)
> dsres <- ds_test(y, "pnorm", 0, 1, type = "eqp", lambda = 1, rounds = 100)
> dsres <- ds_test(y, "pnorm", 0, 1, type = "eqp", lambda = 1)
```

`type="ds"` (default) corresponds to use `ds_1` in one-sample test and `type="eqp"` corresponds to use `ds_eqp_1` in one-sample test. Argument `alpha` is required for `type="ds"` (`ds_1`). To specify a null distribution, we need to give a valid cumulative disbribution function name and its corresponded valid number of parameters. Please refer to `Distributions` in package `stats` for more information. Above examples used the cumulative distribution function of normal distribution. The criterion is the same as K-sample test, *i.e.*, we reject the null hypothesis if `ds_test` gives a DS-statistic larger than 0 (or a preassigned small positive value, 1e-6 for instance).

Here are examples for using `ds_1` and `ds_eqp_1`:

```
> n <- 100
> mu <- 0.5
> x <- rnorm(n, mu, 1)
> y <- pnorm(sort(x), 0, 1)
> lambda <- 1.0
> alpha <- 1.0
> dsres <- ds_1(y, lambda, alpha)
> dsres <- ds_eqp_1(y, lambda)
```

Both these two functions need to sort samples and map them according to a given null cumulative distribution function. The difference between them is that `ds_eqp_1` considers an equal partition on [0, 1] but `ds_1` does not. Candidate slicing boundaries in `ds_eqp_1` only depend on the total number of samples and are unrelated to sample quantiles. In `ds_1` they are immediately to the left or right of sample quantile so that the additional argument `alpha` is needed to avoid two slicing events occur immediately to both sides of one samples at the same time.

2.5 Gene set analysis

Subramanian *et al.* (2005) introduced gene set enrichment analysis (GSEA) to the aggregate effect of genes in unit of "gene set". Specifically, gene set enrichment analysis attempts to determine whether the distribution of biological phenotypes are different between genes in a gene set and the other genes, which can be formulated as a non-parametric two-sample testing problem.

`dslice` provides functions for doing gene set analysis and loading standard format files for gene set analysis (.cls, .gct, .gmt and .gmt). We demonstrate the use of dynamic slicing method on a well studied data set P53 NCI-60, which is available on the GSEA website (http://www.broadinstitute.org/gsea/) after register. The data files we use are P53.cls, P53.gct and C2.gmt. We also include this data set in `dslice` package. To see data sets in `R` packge, one can use

```
> data()
```

There are three data sets in `dslice`: `gsa_exp`, `gsa_label` and `gsa_set`. We can load data for gene set analysis by typing:

```
> data(gsa_exp)
```

```
> data(gsa_label)
> data(gsa_set)
```

This data set assays 10,100 gene expression levels and consists of 17 normal samples and 33 samples with mutated p53. The C2 gene set contains 308 predefined functional gene sets (with gene set size between 15 and 500).

Function `ds_gsa` is designed for doing gene set analysis by dynamic slicing method. Its usage is:

```
ds_gsa(expdat, geneset, label, generank, ..., lambda = 1, bycol = FALSE, minsize =
15, maxsize = 500, randseed = 11235, rounds = 1000)
```

The first three arguments can be either file path or data loaded by api functions `load_gct`, `load_gmt` and `load_cls`. `generank` could be ither an integer vector of rank of each gene according to some statistic, or a character string naming a function which takes gene expression matrix as input and returns a vector of gene rank. We can genrate our rank list from our own rank function. `...` are parameters of the function specified (as a character string) by `generank`. Here is an example to define function used as `generank`:

```
> fc <- function(mat, label)
+ {
+   d0 <- apply(x[,which(label == 0)], 1, mean)
+   d1 <- apply(x[,which(label == 1)], 1, mean)
+   d <- d1 / d0
+   return(order(d))
+ }
```

`lambda` is the penalty in dynamic slicing. `bycol` indicates whether we shuffling gene rank or sample labels to generate background distribution. `rounds` specifies the total number of permutation, which is related to the resolution of empirical p-values.

Function `export_res` exports the object generated by `ds_gsa` to a file.

3 Dynamic slicing model

In this section, we briefly give the theoretical part of dynamic slicing methods. For more details, please refer to Jiang *et al.* (2015).

3.1 Model and theory under `ds_k` and `ds_eqp_k`

Let's go back to the K-sample hypothesis testing problem that `ds_k` and `ds_eqp_k` deal with:

$$H_0 : \text{the distributions of } Y \text{ given } X = j \ (1 \le j \le K) \text{ are the same}$$
$$\text{v.s. } H_1 : \text{the distributions of } Y \text{ given } X = j \ (1 \le j \le K) \text{ are not the same.}$$

In traditional ways, testing started with analyzing Y grouped by values of X, which is $Y|X = j$. Instead of directly modeling the distribution of Y given X, we model the conditional distribution of X given Y. To group X according to Y, all observations are sorted by their values of Y at first. Then we attempt to group X around the rank list. We call the procedure of grouping X according to Y "slicing".

Under H_0, the conditional distribution of X does not depend on slices of Y and

$$X \sim \text{Multinomial}\left(1, (p_1, \ldots, p_K)\right), \quad \sum_{j=1}^{K} p_j = 1.$$

Under H_1, the distribution of X conditional on slices is given by

$$X|S(Y) = h \sim \text{Multinomial}\left(1, (p_1^{(h)}, \ldots, p_K^{(h)})\right), \quad \sum_{j=1}^{K} p_j^{(h)} = 1.$$

We have

$$p_j = \Pr(X = j) = \sum_{h=1}^{|S|} P(S(Y) = h) p_j^{(h)}, \text{ for } j = 1, \ldots, K.$$

and without loss of generality, we assume that $0 < p_1 \le \ldots \le p_K < 1$.

Given a fixed slicing scheme $S(Y)$, the log-likelihood ratio of H_1 versus H_0 can be written as

$$n\widehat{\text{MI}}\left(X, S(Y)\right) = \sum_{h=1}^{|S|} \sum_{j=1}^{K} n_j^{(h)} \log\left(\frac{n_j^{(h)}}{n^{(h)}}\right) - \sum_{j=1}^{K} n_j \log\left(\frac{n_j}{n}\right), \tag{1}$$

where $\widehat{\text{MI}}\left(X, S(Y)\right)$ is the plug-in estimator of the mutual information between X and $S(Y)$ based on observations $\{(x_i, y_i)\}_{i=1}^n$, n_j is the number of observations with $x_i = j$ $(i = 1, \ldots, n$ and $j = 1, \ldots, K)$, $n^{(h)}$ is the number of observations in slice s_h $(h = 1, \ldots, |S|)$ and $n_j^{(h)}$ is the number of observations with $x_i = j$ and $S(y_i) = h$. In a word, our goal is to test whether $\mathbf{p}^{(h)} = (p_1^{(h)}, \ldots, p_K^{(h)})$ is invariant with respect to h, *i.e.*, slicing strategies. Figure 2 shows the likelihood ratio test in a given slicing strategy.

The choice of the slicing scheme $S(Y)$ is important in detecting the dependence between a pair of X and Y in the K-sample testing problem. As shown by Figure 3, different slicing schemes gives distinct results. What's more, unless the number of slices grows with sample size n, it is possible that $\text{MI}(X, T(Y)) = 0$ when $\text{MI}(X, Y) > 0$ under H_1. On the other hand, if we divide observations into too many slices (in the most extreme case each slice only contains one

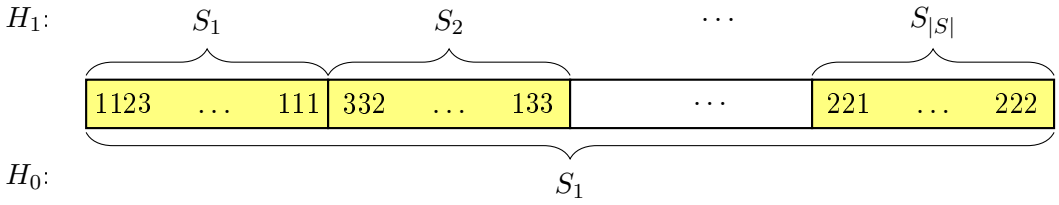

Figure 2: Illustration of likelihood ratio test in dynamic slicing.

observation), there will not enough power to distinguish the alternative and the null hypothesis. To solve this dilemma, we assign a prior on slicing schemes and choose the "optimal" slicing scheme under this prior. The final result is that we use a regularized log-likelihood ratio:

$$n\widehat{\mathcal{D}}_K = \max_S \left[n\widehat{\mathrm{MI}}\left(X, S(Y)\right) - \lambda(n)(|S| - 1) \right], \tag{2}$$

where the maximum is taken over all possible slicing schemes. Furthermore, we assume that the penalty term in (2) takes the form of $\lambda(n) = \lambda_0 \log(n)$, where $\lambda_0 > 0$ is to be specified.

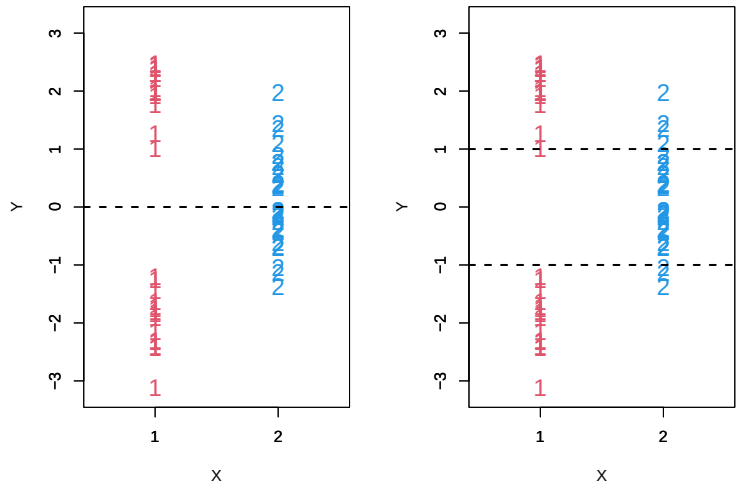

Figure 3: Illustration of different slicing schemes. X is a binary indicator for samples from two populations and Y is a continuous variable. Left panel: by dividing the observations into two slices according to Y, we are not able to reject the null hypothesis since the number of observations with X=1 and X=2 are almost equal in two slices. Right panel: by dividing the observations into three slices according to Y, we can detect the dependence between Y and X.

The searching of optimal slicing strategy follows a dynamic programming procedure, which is a variant of the *Viterbi algorithm*. Under the optimal slicing strategy, Jiang *et al.* (2015) have proofed that $\widehat{\mathcal{D}}_K = 0$ almost surely under the null hypothesis and is larger than a given positive number almost surely under the alternative hypothesis as sample size n goes to infinite.

When n is extremely large, the above procedure may be time consuming, we can first divide ranked observations into $O(n^{1/2})$ bins such that each bin contains approximately $n^{1/2}$ obser-

vations. Then, we can define a test statistic to have the same form as (2) except that the maximization is taken over slicing schemes restricted on the fixed $O(n^{1/2})$ bins (slicing is not allowed within a bin). The corresponding statistics $\widehat{\mathcal{D}}_K^*$ has similar theoretical properties with $\widehat{\mathcal{D}}_K$. For more details, please refer to Jiang *et al.* (2015). Although lower resolution may decrease the power of the test, it can reduce the computational complexity of the dynamic slicing algorithm from $O(n^2)$ to $O(n)$. Figure 4 shows the possible slicing positions (t_i, $i = \{1, 2, 3, 4, 5, 6\}$) in $O(n^{1/2})$-resolution version:

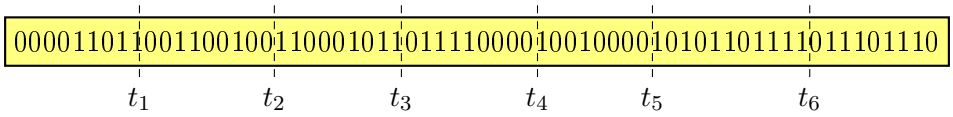

Figure 4: Illustration of possible slicing positions of `ds_eqp_k`

3.2 Model and theory under `ds_1` and `ds_eqp_1`

One-sample testing problem differs from the K-sample problem in our setup, as it can no longer be recast as a test of independence. But a similar slicing idea may apply. As we map the observations to the given cumulative distribution function, the hypothesis testing problem is converted into testing whether their corresponding quantiles are uniformly distributed on [0, 1].

Given a fixed slicing scheme $S(Y)$, the log-likelihood ratio of alternative versus null can be written as

$$n\mathrm{KL}\left(\widehat{P}_n(S(Y))||P_0(S(Y))\right) = \sum_{h=1}^{|S|} n_h \log\left(\frac{n_h}{nw_h}\right), \tag{3}$$

where $w_h = |s_h|$ is the width of slice s_h and n_h is the number of observations in slice s_h. $\mathrm{KL}\left(\widehat{P}_n(S(Y))||P_0(S(Y))\right)$ is the Kullback-Leibler divergence between the empirical distribution of $S(Y)$, $\widehat{P}_n(S(Y))$, and the null distribution of $S(Y)$, $P_0(S(Y))$.

To avoid having too many slices or slices that are arbitrarily small, we introduce a prior on slicing schemes. It gives us the following statistic:

$$n\widehat{\mathcal{D}}_1(\alpha_0) = \max_S \left[n\mathrm{KL}\left(\widehat{P}_n(S(Y))||P_0(S(Y))\right) - \lambda(n)(|S|-1) + \alpha_0 \sum_{h=1}^{|S|} \log(w_h)\right], \tag{4}$$

where the penalty term $\lambda(n) = -\log(\theta_n)$ and is assumed to take the form of $\lambda(n) = \lambda_0 \log(n)$. The parameter λ_0 penalizes the number of slices, while the parameter α_0 penalizes both the width and the number of slices. A larger value of α_0 gives rise to a heavier penalization on small slices and α_0 can be viewed as a prior on the "smoothness" of the estimated densities under the alternative hypothesis.

An variant version of the above procedure is to divide interval [0, 1] into n equal size slices, which avoid the arbitrarily small slices (slicing immediately to the both sides of sample at the

same time). The candidate slicing positions are only depends on the sample size. Let's denote the statistic of this version as $\widehat{\mathcal{D}}_1^*(0)$.

As K-sample test, the optimal slicing strategy could be obtained via a dynamic programming procedure. Under the optimal slicing strategy, Jiang *et al.* (2015) have proofed that $\widehat{\mathcal{D}}_1(\alpha_0)$ and $\widehat{\mathcal{D}}_1^*(0)$ have similar theoretical properties with $\widehat{\mathcal{D}}_K$ when n goes to infinite. For more details, please refer to Jiang *et al.* (2015).

References

Eddelbuettel D, François R, Allaire J, Chambers J, Bates D, Ushey K (2011). "Rcpp: Seamless R and C++ integration." *Journal of Statistical Software*, **40**(8), 1–18.

Jiang B, Ye C, Liu JS (2015). "Non-parametric K-sample Tests via Dynamic Slicing." *Journal of the American Statistical Association*, **110**(510), 642–653.

Subramanian A, Tamayo P, Mootha VK, Mukherjee S, Ebert BL, Gillette MA, Paulovich A, Pomeroy SL, Golub TR, Lander ES, *et al.* (2005). "Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles." *Proceedings of the National Academy of Sciences of the United States of America*, **102**(43), 15545–15550.