Transfer learning for high-dimensional linear regression Notes

19331053 纪传宇

2022 年 7 月 15 日

1 INTRODUCTION

(1)Regression analysis is one of the most widely used statistical methods to understand the association of an outcome with a set of covariates In this paper, we consider transfer learning in high-dimensional linear models. Formally, the target model can be written as

$$y_i^{(0)} = (x_i^{(0)})^T \beta + \epsilon_i^{(0)}, i = 1, ..., n_0 \tag{1}$$

where $((x_i^{(0)})^T, y_i^{(0)}), i = 1, ...n_0$ are independent samples,$\beta \in R^p$ is the coefficient vector of interest,and $\epsilon_i^{(0)}, i = 1, ..., n_0$ are independently distributed random noises with $E[\epsilon_i^{(0)}|x_i^{(0)}] = 0$. In the high-dimensional regime, where p can be larger and much larger than n_0, β is often assumed to be sparse such that the number of nonzero elements of β, denoted by s, is much smaller than p.

(2)In the context of transfer learning, we observe additional samples from K auxiliary studies, That is, we observe $((x_i^{(k)})^T, y_i^{(k)})$ generated from the auxiliary model

$$y_i^{(k)} = (x_i^{(k)})^T w^{(k)} + \epsilon_i^{(k)}, i = 1, ..., n_k, k = 1, ..., K \tag{2}$$

where $w^{(k)} \in R^p$ is the regression vector for the kth study, and $\epsilon_i^{(k)}$is the random noise such that $E[\epsilon_i^{(k)}|x_i^{(k)}] = 0$. The regression coefficients $w^{(k)}$ are unknown and different from our target β in general. The number of auxiliary studies, K, is allowed to grow but practically K may not be too large. We will study the estimation and prediction of target model (1) utilizing the primary data $y_i^{(0)} = (x_i^{(0)})^T \beta + \epsilon_i^{(0)}, i = 1, ..., n_0$as well as the data from K auxiliary studies $y_i^{(k)} = (x_i^{(k)})^T w^{(k)} + \epsilon_i^{(k)}, i = 1, ..., n_k, k = 1, ..., K$

If an auxiliary model is ‘similar’ to the target model, we say that this auxiliary sample/study is informative. In this work, we characterize the informative level of the kth auxiliary study using the sparsity of the difference between $w^{(k)} and \beta$. Let $\delta^{(k)} = \beta - w^{(k)}$ denote the contrast between $w^{(k)} and \beta$. The set of informative auxiliary samples is those whose contrasts are sufficiently sparse:

$$\mathcal{A}_q = \left\{1 \le k \le K : \left\|\delta^{(k)}\right\|_q \le h\right\} \tag{3}$$

for some $q \in [0, 1]$. The set A_q contains the auxiliary studies whose contrast vectors have ℓ_q-sparsity at most h and is called the informative set. It will be seen later that as long as h is relatively small compared to the sparsity of β, the studies in A_q can be useful in improving the prediction and estimation of β For any $q \in [0, 1]$, smaller h implies that the auxiliary samples in $\mathcal{A}_q$ are more informative; larger cardinality of $\mathcal{A}_q$ ($|\mathcal{A}_q|$) implies that a larger number of informative auxiliary samples. Therefore, smaller h and larger $|\mathcal{A}_q|$ should be favourable. We allow $\mathcal{A}_q$ to be empty in which case none of the auxiliary samples is informative. For the auxiliary samples outside of $\mathcal{A}_q$, we do not assume sparse $\delta^{(k)}$ and hence $w^{(k)}$ can be very different from β for $k \notin \mathcal{A}_q$.

2 Notation

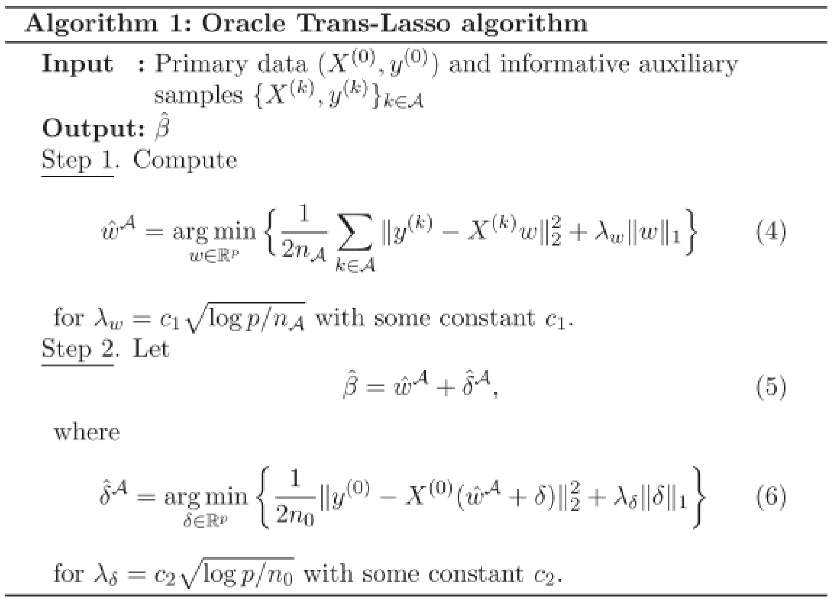
2.1 title

表 **1:** Notation for Trans-Lasso algorithm

Indices	
$X^{(0)} \in R^{n_0 * p}$	design matirx for the primary data
$Y^{(0)} \in R^{n_0}$	response vextor for the primary data
$X^{(k)} \in R^{n_k * p}$	design matrix for the kth auxiliary data
$Y^{(k)} \in R^{n_k}$	the response vector for the kth auxiliary data
$\{R_l\}_{l \in \mathcal{L}}$	$R_l, l \in \mathcal{L}$
$n_{\mathcal{A}_q}$	$\sum_{k \in \mathcal{A}_q} n_k$
$\Lambda_{\max}(\Sigma)$	largest eigenvalues of Σ respectively
$\Lambda_{\min}(\Sigma)$	smallest eigenvalues of Σ respectively
$a \vee b$	$\max\{a, b\}$
$a \wedge b$	$\min\{a, b\}$
$a_n = O\left(b_n\right)$ and $a_n \lesssim b_n$	$\|a_n / b_n\| \leq c$for some constant c when n is large enough
$a_n \asymp b_n$	$\|a_n / b_n\| \rightarrow c$ for some constant c as $n \rightarrow \infty$
$a_n = O_P\left(b_n\right)$ and $a_n \lesssim_{\mathbb{P}} b_n$	$\mathbb{P}\left(\|a_n / b_n\| \leq c\right) \rightarrow 1$ for some constant $c < \infty$
$a_n = o_P\left(b_n\right)$	$(\|an/bn\| > c) \rightarrow 0 for any constant c > \ 0$

3 ESTIMATION WITH KNOWN INFORMATIVE AUXILIARY SAMPLES

3.1 Oracle Trans-Lasso algorithm



```
Algorithm 1: Oracle Trans-Lasso algorithm
Input  : Primary data $(X^{(0)}, y^{(0)})$ and informative auxiliary
         samples $\{X^{(k)}, y^{(k)}\}_{k\in\mathcal{A}}$
Output: $\hat{\beta}$
Step 1. Compute
    $\hat{w}^{\mathcal{A}} = \arg\min_{w\in\mathbb{R}^p} \left\{ \frac{1}{2n_{\mathcal{A}}} \sum_{k\in\mathcal{A}} \|y^{(k)} - X^{(k)}w\|_2^2 + \lambda_w\|w\|_1 \right\}$   (4)
  for $\lambda_w = c_1\sqrt{\log p/n_{\mathcal{A}}}$ with some constant $c_1$.
Step 2. Let
    $\hat{\beta} = \hat{w}^{\mathcal{A}} + \hat{\delta}^{\mathcal{A}},$   (5)
  where
    $\hat{\delta}^{\mathcal{A}} = \arg\min_{\delta\in\mathbb{R}^p} \left\{ \frac{1}{2n_0} \|y^{(0)} - X^{(0)}(\hat{w}^{\mathcal{A}} + \delta)\|_2^2 + \lambda_\delta\|\delta\|_1 \right\}$   (6)
  for $\lambda_\delta = c_2\sqrt{\log p/n_0}$ with some constant $c_2$.
```

Fig. 1. algorithm of Oracle Trans-Lasso algorithm

3.2 step of Oracle Trans-Lasso algorithm

$\mathcal{A}$ is known in this algorithm

(1) compute an initial estimator using all the informative auxiliary samples $\widehat{w}^{\mathcal{A}}$is realized based on the Lasso (Tibshirani, 1996) using all the informative auxiliary samples

to find $\hat{w}^{\mathcal{A}}$ in

$$\hat{w}^{\mathcal{A}} = \arg\min_{w\in\mathbb{R}^p} \left\{ \frac{1}{2n_{\mathcal{A}}} \sum_{k\in\mathcal{A}} \left\|y^{(k)} - X^{(k)}w\right\|_2^2 + \lambda_w\|w\|_1 \right\} \quad (4)$$

is equivalent to find

$$\hat{w}^{\mathcal{A}} = \underset{w \in \mathbb{R}^p}{\arg\min} \left\{ \frac{1}{2n_{\mathcal{A}}} \sum_{k \in \mathcal{A}} w^T x^{(k)T} x^{(k)} w - 2y^{(k)T} x^{(k)} w + y^{(k)T} y \right\}$$

(5)

let f(w) denote

$$\left\{ \frac{1}{2n_{\mathcal{A}}} \sum_{k \in \mathcal{A}} w^T x^{(k)T} x^{(k)} w - 2y^{(k)T} x^{(k)} w + y^{(k)T} y \right\} \tag{6}$$

then we can use maximum likelihood estimates to evaluate $\hat{w}^{\mathcal{A}}$ if we plug $^{\mathcal{A}}$ into the formula we can get E[$\frac{\partial f(w)}{\partial w}$ $|_{\hat{w}^{A}}$]=0 which is equal to

$$E\left[\frac{1}{n_A}\sum x^{(k)^\top}\left(x^{(k)}w - y^{(k)}\right)\right] = 0 \tag{7}$$

so the equation in the article

$$\mathbb{E}\left[\sum_{k \in \mathcal{A}} \left(X^{(k)}\right)^{\mathrm{T}} \left(y^{(k)} - X^{(k)} w^{\mathcal{A}}\right)\right] = 0 \tag{8}$$

can be proved

Denoting $\mathbb{E}\left[x_i^{(k)}\left(x_i^{(k)}\right)^\top\right] = \Sigma^{(k)}$, $w^{\mathcal{A}}$has the following explicit form:

$$w^{\mathcal{A}} = \beta + \delta^{\mathcal{A}} \tag{9}$$

the probabilistic limit of $\widehat{\boldsymbol{W}}^{\mathcal{A}}$,has bias $\delta^{\mathcal{A}}$,and $\delta^{\mathcal{A}}$ can be calculated by

$$\delta^{\mathcal{A}} = \sum_{k \in \mathcal{A}} n_k / n_{\mathcal{A}} \times \delta^{(k)} \tag{10}$$

(2) Correct its bias using the primary data in the second step $\delta^{\mathcal{A}}$is a sparse high-dimensinal vector while ℓ_1- norm is no larger than h.

Because we set $\mathcal{A}_q = \left\{1 \leq k \leq K : \left\|\delta^{(k)}\right\|_q \leq h\right\}$ and in this algorithm q is 1. What's more, $\left\|\delta^{(k)}\right\|_1 \leq h$ and $\delta^{\mathcal{A}} = \sum_{k \in \mathcal{A}} n_k / n_{\mathcal{A}} \times \delta^{(k)}$.

Hence, the error of step 2 is under control for a relatively small h

The Oracle Trans-Lasso does not penalize the differences among the regression coefficients in the auxiliary studies. This is again because the focus of transfer learning is only the target study. Theoretically, extra penalization terms and the joint analysis of multiple estimators may not help improve the estimation accuracy of the parameter of interest.

4 UNKNOWN SET OF INFORMATIVE AUXILIARY SAMPLES

we propose a data-driven method for estimation and prediction when $\mathcal{A}$ is unknown.

4.1 The Trans-Lasso algorithm

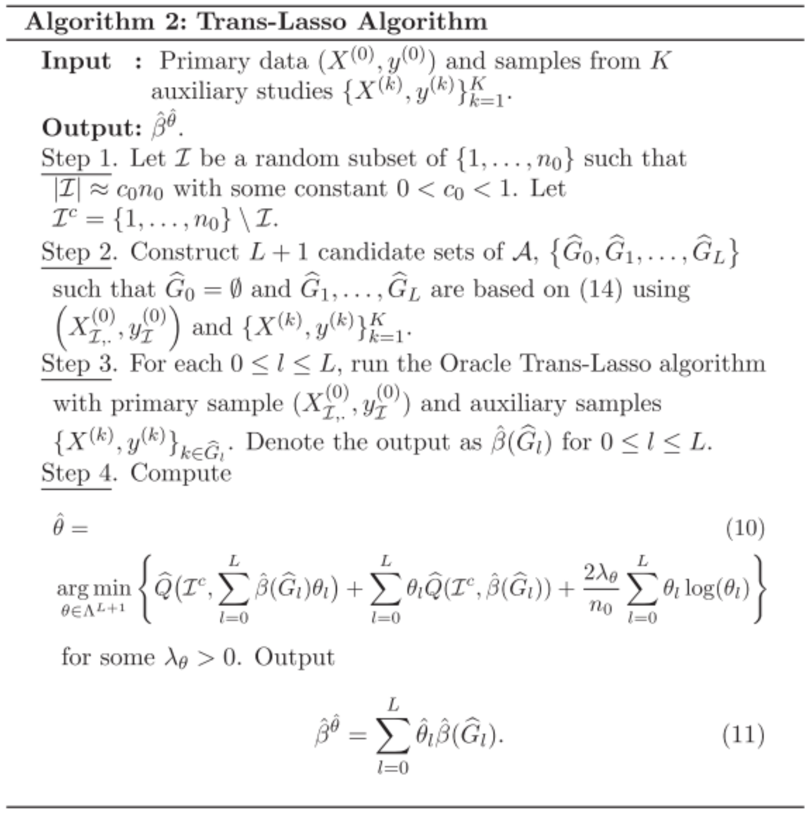
two main step: (1) construct a collection of candidate estimators, each of which is based on an estimate of $\mathcal{A}$.

(2) perform an aggregation step on these candidate estimators

more notation: For a generic estimate of , b, denoteits sum of squared prediction error as

$$\widehat{Q}(\mathcal{I}, b) = \sum_{i \in \mathcal{I}} \left\| y_i^{(0)} - \left(x_i^{(0)} \right)^\top b \right\|_2^2 \tag{11}$$

$\mathcal{I}$ is a subset of $\{1, \ldots, n_0\}$. Let $\Lambda^{L+1} = \left\{ v \in \mathbb{R}^{L+1} : v_l \geq 0, \sum_{l=0}^{L} v_l = 1 \right\}$denote an L-dimensional simplex



Algorithm 2: Trans-Lasso Algorithm

```
Input  : Primary data $(X^{(0)}, y^{(0)})$ and samples from $K$
         auxiliary studies $\{X^{(k)}, y^{(k)}\}_{k=1}^{K}$.
Output: $\hat{\beta}^{\hat{\theta}}$.
Step 1. Let $\mathcal{I}$ be a random subset of $\{1, \ldots, n_0\}$ such that
    $|\mathcal{I}| \approx c_0 n_0$ with some constant $0 < c_0 < 1$. Let
    $\mathcal{I}^c = \{1, \ldots, n_0\} \setminus \mathcal{I}$.
Step 2. Construct $L+1$ candidate sets of $\mathcal{A}$, $\{\widehat{G}_0, \widehat{G}_1, \ldots, \widehat{G}_L\}$
    such that $\widehat{G}_0 = \emptyset$ and $\widehat{G}_1, \ldots, \widehat{G}_L$ are based on (14) using
    $\left(X^{(0)}_{\mathcal{I},\cdot}, y^{(0)}_{\mathcal{I}}\right)$ and $\{X^{(k)}, y^{(k)}\}_{k=1}^{K}$.
Step 3. For each $0 \leq l \leq L$, run the Oracle Trans-Lasso algorithm
    with primary sample $(X^{(0)}_{\mathcal{I},\cdot}, y^{(0)}_{\mathcal{I}})$ and auxiliary samples
    $\{X^{(k)}, y^{(k)}\}_{k \in \widehat{G}_l}$. Denote the output as $\hat{\beta}(\widehat{G}_l)$ for $0 \leq l \leq L$.
Step 4. Compute
    $\hat{\theta} = \underset{\theta \in \Lambda^{L+1}}{\arg\min} \left\{ \widehat{Q}(\mathcal{I}^c, \sum_{l=0}^{L} \hat{\beta}(\widehat{G}_l)\theta_l) + \sum_{l=0}^{L} \theta_l \widehat{Q}(\mathcal{I}^c, \hat{\beta}(\widehat{G}_l)) + \frac{2\lambda_\theta}{n_0} \sum_{l=0}^{L} \theta_l \log(\theta_l) \right\}$    (10)
    for some $\lambda_\theta > 0$. Output
    $\hat{\beta}^{\hat{\theta}} = \sum_{l=0}^{L} \hat{\theta}_l \hat{\beta}(\widehat{G}_l)$.    (11)
```

Fig. 2. algorithm of Trans-Lasso algorithm

steps 2 and 3 of the Trans-Lasso algorithm construct some initial estimates of $\beta, \widehat{\beta}\left(\widehat{G}_l\right)$ They are computed using the Oracle Trans-Lasso algorithm by treating eachˆ$\widehat{G}_l$ as the set of informative auxiliary samples.

Step 4 is based on the Q-aggregation proposed in Dai et al. (2012) with a uniform prior, a Kullback - Leibler penalty, and a simplified tuning parameter. The Q-aggregation can be viewed as a weighted version of least square aggregation and exponential aggregation (Rigollet Tsybakov, 2011) and it has been shown to be rate optimal both in expectation and with high probability for model selection aggregation problems

4.2 Constructing the candidate sets for aggregation

Model selection aggregation is an effective method for the transfer learning task under consideration

let us first point out a naive construction of candidate sets, which consists of 2^k candidates. These candidates are all different combinations of 1, ⋯ , Kdenoted by $\widehat{G}_1, \ldots, \widehat{G}_{2^K}$.$\mathcal{A}$ is an element of these candidate sets.However, the number of candidates is too large and it can be computationally burdensome.In contrast, we would like to pursue a much smaller number of candidate sets such that the cost of aggregation is almost negligible and

$$\mathbb{P}\left(\widehat{G}_l \subseteq \mathcal{A}, \text{ for some } 1 \leq l \leq L\right) \rightarrow 1 \tag{12}$$

can be achieved under mild conditions.

the ideas to solve this problem is to is to exploit the sparsity patterns of the contrast vectors.The definition of $\mathcal{A}$ implies that $\left\{\delta^{(k)}\right\}_{k \in \mathcal{A}}$ are sparser than $\left\{\delta^{(k)}\right\}_{k \in \mathcal{A}^c}$,where $\mathcal{A}^c = \{1, \ldots, K\} \backslash \mathcal{A}$ This property motivates us to find a sparsity index $R^{(k)}$ and its estimator $\widehat{R}^{(k)}$ for each $1 \leq k \leq K$ such that

$$\max_{k \in \mathcal{A}^o} R^{(k)} < \min_{k \in \mathcal{A}^c} R^{(k)} \quad \text{ and } \quad \mathbb{P}\left(\max_{k \in \mathcal{A}^o} \widehat{R}^{(k)} < \min_{k \in \mathcal{A}^c} \widehat{R}^{(k)}\right) \rightarrow 1, \tag{13}$$

where $\mathcal{A}^0$ is some subset of $\mathcal{A}$. In words, the sparsity indices in o are no larger than the sparsity indices in $\mathcal{A}^c$ and so are their estimators with high probability. To utilize Equation (13), we can define the candidate sets as

$$\widehat{G}_l = \left\{1 \leq k \leq K : \widehat{R}^{(k)} \text{ is among the first } l \text{ smallest of all }\right\} \tag{14}$$

for $1 \leq l \leq K$. That is, $\widehat{G}_l$ is the set of auxiliary samples whose estimated sparsity indices are among the first l smallest. A direct consequence of Equations (13) and (14) is that $\mathbb{P}\left(\widehat{G}_{|\mathcal{A}^o|} = \mathcal{A}^o\right) \rightarrow 1$ and

To achieve the largest gain with transfer learning, we would like to find proper sparsity indices such that Equation (13) holds for $\sum_{k \in \mathcal{A}^o} n_k$ as large as possible. Notice that $\widehat{G}_{K+1} = \{1, \ldots, K\}$ is always included as candidates according to Equation (14). Hence, in the special cases where all the auxiliary samples are informative or none of the auxiliary samples are informative, it holds that $\widehat{G}_{|\mathcal{A}|} = \mathcal{A}$ and the Trans-Lasso is not much worse than the Oracle Trans-Lasso. The more challenging cases are $0 < |\mathcal{A}| < K$.

As $\left\{\delta^{(k)}\right\}_{k\in\mathcal{A}^c}$ are not necessarily sparse, the estimation of $\delta^{(k)}$ or functions of $\delta^{(k)}$, $1 \leq k \leq K$, is not trivial. As an example, an intuitive sparsity index can be $\left\|\delta^{(k)}\right\|_1$ and its estimate is $\left\|\widehat{\beta}\left(\widehat{G}_0\right) - \widehat{w}^{(k)}\right\|_1$, where $\widehat{w}^{(k)}$ is the Lasso estimate of $w^{(k)}$ based on the k th study. However, such a Lasso-based estimate is not guaranteed to converge to the oracle $\left\|\delta^{(k)}\right\|_1$ when $\delta^{(k)}$ is non-sparse. Therefore, we consider using $R^{(k)} = \left\|\Sigma\delta^{(k)}\right\|_2^2$, which is a function of the population-level marginal statistics, as the oracle sparsity index for k th auxiliary sample. The advantage of $R^{(k)}$ is that it has a natural unbiased estimate even when $\delta^{(k)}$ is non-sparse. Let us relate $R^{(k)}$ to the sparsity of $\delta^{(k)}$ using a Bayesian characterization of sparse vectors assuming $\Sigma^{(k)} = \Sigma$ for all $0 \leq k \leq K$. If $\delta_j^{(k)}$ are i.i.d. Laplacian distributed with mean zero and variance v_k^2 for each k , then it follows from the properties of Laplacian distribution (Liu & Kozubowski, 2015) that $\mathbb{E}\left[\left\|\delta^{(k)}\right\|_1\right] \asymp \mathbb{E}^{1/2}\left[\left\|\Sigma\delta^{(k)}\right\|_2^2\right]$. Hence, the rank of $\mathbb{E}\left[\left\|\Sigma\delta^{(k)}\right\|_2^2\right]$ is the same as the rank of $\mathbb{E}\left[\left\|\delta^{(k)}\right\|_1\right]$. As $\max_{k\in\mathcal{A}}\left\|\delta^{(k)}\right\|_1 < \min_{k\in\mathcal{A}^c}\left\|\delta^{(k)}\right\|_1$, it is reasonable to expect $\max_{k\in\mathcal{A}}\left\|\Sigma\delta^{(k)}\right\|_2^2 < \min_{k\in\mathcal{A}^c}\left\|\Sigma\delta^{(k)}\right\|_2^2$. The above derivation holds for many other zero mean prior distributions besides Laplacian. This illustrates our motivation for considering $R^{(k)}$ as the oracle sparsity index.

We next introduce the estimated version $\widehat{R}^{(k)}$, based on the primary data $\left\{\left(x_i^{(0)}\right)^\top, y_i^{(0)}\right\}_{i\in\mathcal{I}}$ (after sample splitting) and auxiliary samples $\left\{X^{(k)}, y^{(k)}\right\}_{k=1}^K$. We first perform a SURE screening (Fan & Lv, 2008) on the marginal statistics to reduce the effects of random noises.

In this article,if we have the linear regression:

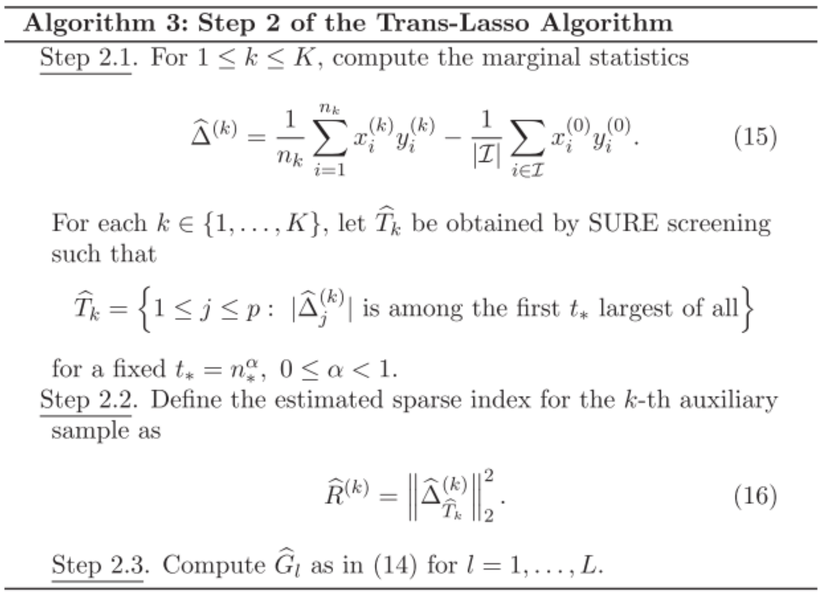
$$y_i = \beta_0 + x_{1i}^\top\beta_1 + \cdots + x_{pi}^\top\beta_p + \varepsilon_i, \quad i = 1, \ldots, n \tag{15}$$

when the number of variables p is much greater than n,how to do with this data.Fan and Lv consider that the marginal correlation coefficient of x_{ij}andy_j that corr $(y_i, x_{ij}) := \omega_j$ should be highly correlated with the regression coefficient β_j

So if β_j is small, the marginal correlation coefficient of x_{ij}andy_j should be very small in most cases.So before using Lasso in variable selection for an ultra-high dimensional regression model, we should use some measures related to marginal correlation coefficient to rank the marginal correlation coefficients.And only the covariates corresponding to the top $t_* = n_*^\alpha$ correlation coefficients (absolute values of) are selected and put into the regression model

We summarize our proposal for Step 2 of the Trans-Lasso as follows (Algorithm 3). Let $n_* = \min_{0 \leq k \leq K} n_k$.

One can see that $\widehat{\Delta}^{(k)}$ are empirical marginal statistics such that $\mathbb{E}\left[\widehat{\Delta}^{(k)}\right] = \Sigma\delta^{(k)}$ for $k \in \mathcal{A}$. The set $\widehat{T}_k$ is the set of first t_* largest marginal statistics for the k th sample. The purpose of screening the marginal statistics is to reduce the magnitude of noise. Notice that the un-screened version $\left\|\widehat{\Delta}^{(k)}\right\|_2^2$ is a sum of p random variables and it contains noise of order $p/\left(n_k \wedge n_0\right)$, which diverges fast as p is much larger than the sample sizes. By screening with t_* of order $n_*^{\alpha, \alpha<1}$, the errors induced by the random noises is under control. In practice, the auxiliary samples with very small sample sizes can be removed from the analysis as their contributions to the target problem is mild. Desirable choices of $\widehat{T}_k$ should keep the variation of $\Sigma\delta^{(k)}$ as much as possible. Under proper conditions, SURE screening can consistently select a set of strong marginal statistics and hence is appropriate for the current purpose. In Step 2.2, we compute $\widehat{R}^{(k)}$ based on the marginal statistics which are selected by SURE screening. In practice, different choices of t_* may lead to different realizations of $\widehat{G}_l$. One can compute multiple sets of $\left\{\widehat{R}^{(k)}\right\}_{k=1}^K$ with different t_* which give multiple sets of $\left\{\widehat{G}_l\right\}_{l=1}^K$. It will be seen from Lemma 1 that a finite number of choices on t_* does not affect the rate of convergence.



Algorithm 3: Step 2 of the Trans-Lasso Algorithm

Step 2.1. For $1 \leq k \leq K$, compute the marginal statistics

$$\widehat{\Delta}^{(k)} = \frac{1}{n_k} \sum_{i=1}^{n_k} x_i^{(k)} y_i^{(k)} - \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} x_i^{(0)} y_i^{(0)}. \tag{15}$$

For each $k \in \{1, \ldots, K\}$, let $\widehat{T}_k$ be obtained by SURE screening such that

$$\widehat{T}_k = \left\{ 1 \leq j \leq p : \ |\widehat{\Delta}_j^{(k)}| \text{ is among the first } t_* \text{ largest of all} \right\}$$

for a fixed $t_* = n_*^{\alpha}, \ 0 \leq \alpha < 1$.

Step 2.2. Define the estimated sparse index for the k-th auxiliary sample as

$$\widehat{R}^{(k)} = \left\| \widehat{\Delta}_{\widehat{T}_k}^{(k)} \right\|_2^2. \tag{16}$$

Step 2.3. Compute $\widehat{G}_l$ as in (14) for $l = 1, \ldots, L$.

Fig. 3. step 2 of the Trans-Lasso algorithm

4.3 Theoretical properties of Trans-Lasso

we first establish the moedl selection aggregation type of results for the Trans-Lasso estimator

Lemma 1 (Q-aggregation for Trans-Lasso). Assume that Conditions 1 and 2 hold true. Let $\hat{\theta}$ be computed via Equation (10) with $\lambda_\theta \geq 4\sigma_0^2$. With probability at least $1 -$ t , it holds that

$$\frac{1}{|\mathcal{I}^c|} \left\| X_{\mathcal{I}^c,.,}^{(0)} (\widehat{\beta} - \beta) \right\|_2^2 \leq \min_{0 \leq l \leq L} \frac{1}{|\mathcal{I}^c|} \left\| X_{\mathcal{I}^c,.}^{(0)} \left(\widehat{\beta}\left(\widehat{G}_l\right) - \beta \right) \right\|_2^2 + \frac{\lambda_\theta \log(L/t)}{n_0}. \tag{16}$$

If $L \leq c_1 n_0$ for some small enough constant c_1 , then

$$\|\widehat{\beta}^{\widehat{\theta}} - \beta\|_2^2 \lesssim_{\mathbb{P}} \min_{0 \leq l \leq L} \left\| \widehat{\beta}\left(\widehat{G}_l\right) - \beta \right\|_2^2 + \frac{\log L}{n_0} \tag{17}$$

Lemma 1 implies that the performance of $\widehat{\beta}^{\widehat{\theta}}$ only depends on the best candidate regardless of the performance of other candidates under mild conditions.As the original Lasso is always in our dictionary, Equations (16) and (17) imply that $\hat{\ } \widehat{\beta}^{\widehat{\theta}}$ is not much worse than the Lasso in prediction and estimation. Formally, 'not much worse' refers to the last term in Equation (16), which can be viewed as the cost of 'searching' for the best candidate model within the dictionary which is of order $\log L/n_0$. Thisterm is almost negligible,say, when L = O(K), which corresponds to our constructed candidate estimators. This demonstrates the robustness of $\widehat{\beta}^{\widehat{\theta}}$ to adversarial auxiliary samples.

Indeed, an estimator with ℓ_2-error guarantee is crucial for more challenging tasks, such as out-of-sample prediction and inference. For our transfer learning task, we show in Equation (17) that the estimation error is of the same order if the cardinality of the dictionary is $L \leq cn_0$ for some small enough c. For our constructed dictionary, it suffices to require $K \leq cn_0$. In many practical applications, K is relatively small compared to the sample sizes and hence this assumption is not very restrictive.

For each $k \in \mathcal{A}^c$, define a set

$$H_k = \left\{1 \leq j \leq p : \left|\Sigma_{j,.}^{(k)} w^{(k)} - \Sigma_{j,.}^{(0)} \beta\right| > n_*^{-\kappa}, \kappa < \alpha/2\right\}.$$

(18)

Recall that $\alpha < 1$ is defined such that $t_* = n^\alpha$. In fact, H_k is the set of 'strong' marginal statistics that can be consistently selected into $\widehat{T}_k$ for each $k \in \mathcal{A}^c$. We see that $\Sigma_{j,.}^{(k)} w^{(k)} - \Sigma_{j,.}^{(0)} \beta = \Sigma_{j,.} \delta^{(k)}$ if $\Sigma^{(k)} = \Sigma^{(0)}$ for $k \in \mathcal{A}^c$. The definition of $\mathcal{H}_k$ in Equation (18) allows for heterogeneous designs among non-informative auxiliary samples.

5 SIMULATION STUDIES

we evaluate the empirical performance of the proposed methods and some other comparable methods in various numerical experiments. Specifically, we evaluate the performance of five methods, including Lasso, Oracle Trans-Lasso proposed in Section 2.1, TransLasso proposed in Section 3.1, and two other ad hoc transfer learning methods related to ours

(1)'Lasso'

(2)'Oracle Trans-Lasso'

(3)'Trans-Lasso'

(4)'aggregated Lasso', which is Trans-Lasso except the bias-correction step (Step 2) of the Oracle Trans-Lasso

(5)'ad hoc ℓ_1-transfer', which follows the steps of Trans-Lasso but uses a different aggregation step.Considering $\widehat{R}^{(k)} = \left\|\widehat{\beta}^L - \widehat{w}^{(k)}\right\|_1, k = 1, \ldots, K$, where $\widehat{\beta}^L$ and $\widehat{w}^{(k)}$ are the Lasso estimators based on each of the corresponding studies. Moreover, the Q-aggregation step is replaced with the cross-validation, where we select the set $\widehat{G}_l$ that minimizes the out-of-sample prediction errors.

5.1 Retrieval results with identity covariance matrix for the designs

We consider p=500,$n_0 = 150$, and $n_1, \ldots, n_K = 100$ for K=20 . The covariates $x_i^{(k)}$ are i.i.d. Gaussian with mean zero and identity covariance matrix for all $0 \leq k \leq K$ and $\epsilon_i^{(k)}$ are i.i.d. Gaussian with mean zero and variance one for all $0 \leq k \leq K$. For the target parameter β , we set s=16, $\beta_j = 0.3$ for $j \in \{1, \ldots, s\}$, and $\beta_j = 0$ otherwise. For the regression coefficients in auxiliary samples, we consider two configurations.

(i) For a given $\mathcal{A}$, if $k \in \mathcal{A}$, let

$$w_j^{(k)} = \beta_j - 0.31\,(j \in H_k) \tag{19}$$

where H_k is a random subset of [p] with $|H_k| = h \in \{2, 6, 12\}$. If k $\notin \mathcal{A}$, we set H_k to be a random subset of [p] with $|H_k| = 2s$ and $w_j^{(k)} = \beta_j - 0.51\,(j \in H_k)$. We set $w_1^{(k)} = -0.3$ for $k = 1, \ldots, K$.

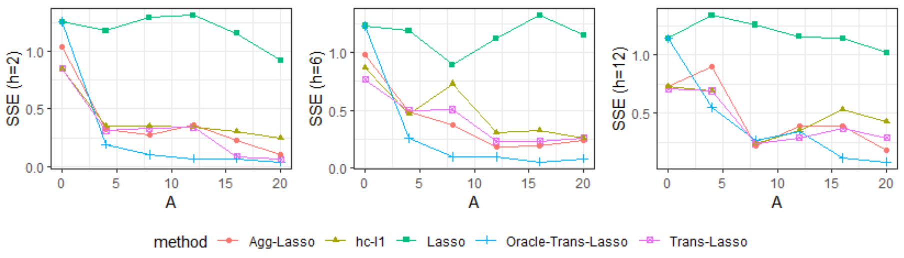


Fig. 4. Estimation errors of the ad hoc ℓ_1-transfer, Agg-Lasso, Lasso, Oracle Trans-Lasso, and Trans-Lasso with identity covariance matrices of the predictor in (i)

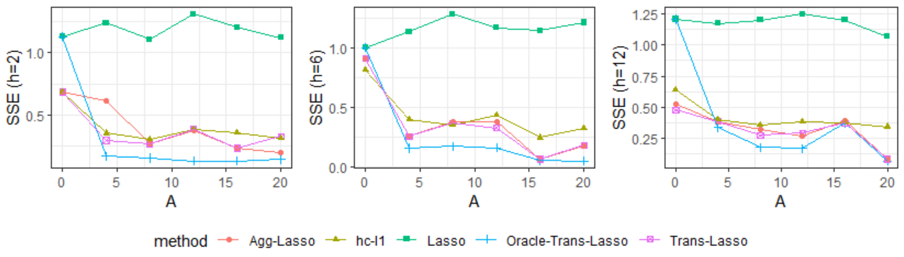


Fig. 5. Estimation errors of the ad hoc ℓ_1-transfer, Agg-Lasso, Lasso, Oracle Trans-Lasso, and Trans-Lasso with identity covariance matrices of the predictor in (ii)

(ii) For a given $\mathcal{A}$, if $k \in \mathcal{A}$, let $H_k = \{1, \dots, 100\}$ and

$$w_j^{(k)} = \beta_j + \xi_j \nVdash (k \in H_k), \quad \text{where } \xi_j \sim_{i.i.d.} N(0, h/100), \tag{20}$$

where $h \in \{2, 6, 12\}$ and $N(a, b)$ is the normal with mean a and standard deviation b . If $k \notin \mathcal{A}$, we set $H_k = \{1, \dots, 100\}$ and

$$w_j^{(k)} = \beta_j + \xi_j 1 (j \in H_k), \quad \text{where } \xi_j \sim_{i.i.d.} N(0, 2s/100) \tag{21}$$

We set $w_1^{(k)} = -0.3$ for $k = 1, \dots, K$. The setting (i) can be treated as either ℓ_0- or ℓ_1- sparse contrasts. In practice, the true parameters are unknown and we use $\mathcal{A}$ to denote the set of auxiliary samples without distinguishing $\ell_0 - or \ell_1$ -sparsity. We consider $|\mathcal{A}| \in \{0, 4, 8, \dots, 20\}$

In the performance,we can clearly see taht the Lasso does not change as $|\mathcal{A}|$ increaes. The other algorithms based on transfer learning have estimation errors decreasing as $|\mathcal{A}|$ increases.In settings (i) and (ii), the Oracle Trans-Lasso has the smallest estimation errors in most settings.In the article the proposed Trans-Lasso is always the second best but in my code repetition, Agg-Lasso almost as same as Trans-Lasso.When $\mathcal{A} = \emptyset$, the Trans-Lasso can have smaller errors than the oracle Trans-Lasso where the latter one does not use auxiliary information. This implies that some auxiliary information can still be borrowed. Due to the randomness of the parameter generation, our definition of $\mathcal{A}$ may not always be the best subset of auxiliary samples that give the smallest estimation errors.

The article mention that Agg-Lasso method has larger estimation errors than Trans-Lasso and ad hoc ℓ_1-transfer. But in my repetition, Agg-Lasso's SSE is not too different from Trans-Lasso.

5.2 Retrieval results with homogeneous designs among $\mathcal{A} \cup \{0\}$

We now consider $x_i^{(k)}$ as i.i.d. Gaussian with mean zero and a equi-correlated covariance matrix, where $\Sigma_{j,j} = 1$ and $\Sigma_{j,k} = 0.8$ if $j \neq k$ for $k \in \mathcal{A} \cup \{0\}$. For $k \notin \mathcal{A} \cup \{0\}$,$x_i^{(k)}$ are i.i.d. Gaussian with mean zero and a Toeplitz covariance matrix whose first row is

$$\Sigma_{1,\cdot}^{(k)} = (1, \underbrace{1/(k+1), \ldots, 1/(k+1)}_{2k-1}, 0_{p-2k}) \tag{22}$$

Other true parameters and the dimensions of the samples are set to be the same as in Section 5.1.

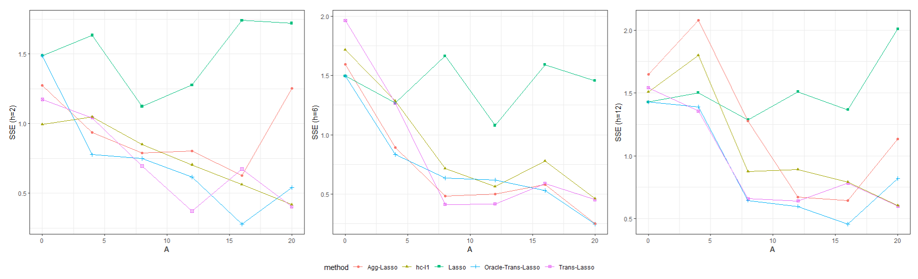


Fig. 6. Estimation errors of the ad hoc ℓ_1-transfer, Agg-Lasso, Lasso, Oracle Trans-Lasso, and Trans-Lasso with identity covariance matrices of the predictor in (i)

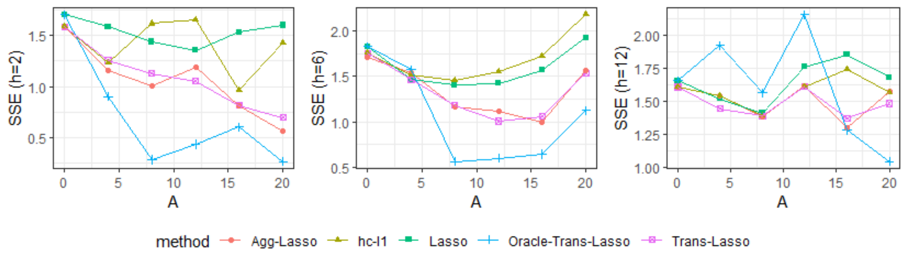


Fig. 7. Estimation errors of the ad hoc ℓ_1-transfer, Agg-Lasso, Lasso, Oracle Trans-Lasso, and Trans-Lasso with identity covariance matrices of the predictor in (ii)

From the results presented in Fig 6 and Fig 7,except (ii)configurations and $|h| = 12$, the Trans-Lasso and Oracle Trans-Lasso have reliable performance in the current setting.However the SSE are large in this type then in Section 5.1 because of the highly correlated covariates. When h is relatively large,accroding to the article Agg-Lasso and ad hoc ℓ_1-transfer have significantly larger estimation errors than Trans-Lasso.However, during repetition,we can see that Agg-Lasso's SSE still is not too different from Trans-Lasso.But the figure also demonstrates the advantage of Trans-Lasso over some ad hoc methods

5.3 Retrieval results with heterogeneous designs

We next consider a setting where $\Sigma^{(k)}$ are distinct for $k = 0, \ldots, K$. Specifically, for $k = 1, \ldots, K, letx_i^{(k)}$ as i.i.d. Gaussian with mean zero and a Toeplitz covariance matrix whose first row is Equation (22). Moreover, $\Sigma^{(0)} = I_p$. Other parameters and the dimensions of the samples are set to be the same as in Section 5.1

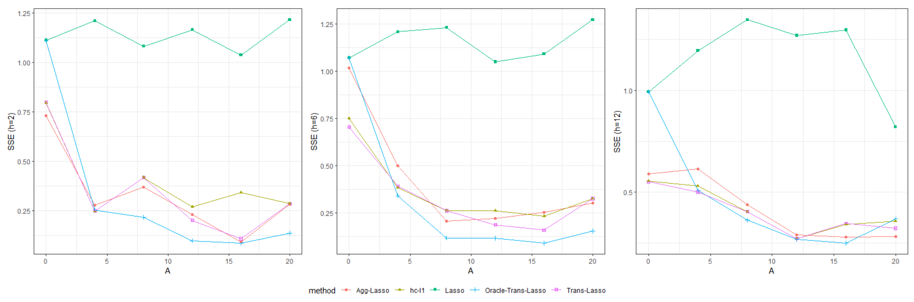


Fig. 8. Estimation errors of the ad hoc ℓ_1-transfer, Agg-Lasso, Lasso, Oracle Trans-Lasso, and Trans-Lasso with identity covariance matrices of the predictor in (i)

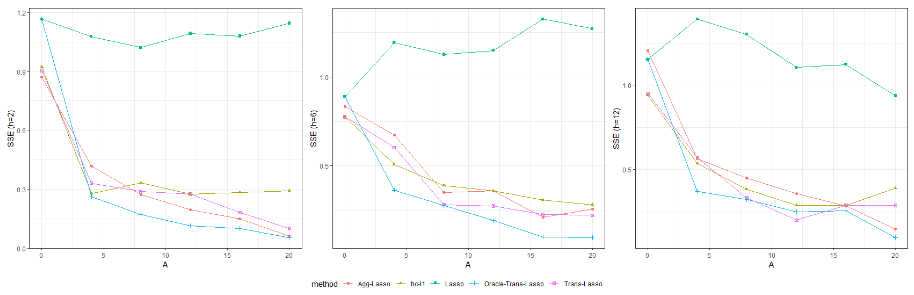


Fig. 9. Estimation errors of the ad hoc ℓ_1-transfer, Agg-Lasso, Lasso, Oracle Trans-Lasso, and Trans-Lasso with identity covariance matrices of the predictor in (ii)

From the results presented in Fig8 and Fig9,the Trans-Lasso and Oracle Trans-Lasso and the Agg-Lasso have reliable performance in the current setting.In some situation,Agg-Lasso's SSE is smller than Trans-Lasso which is different from the article's demonstration

5.4 SIMULATION STUDIES CODE

The R code for all the methods

```

  library(glmnet)
  agg.fun<- function(B, X.test,y.test, total.step=10,
selection=F){
    if(sum(B==0)==ncol(B)*nrow(B)){
      return(rep(0,nrow(B)))
    }
    p<-nrow(B)
    K<-ncol(B)
    colnames(B)<-NULL
    if(selection){#select beta.hat with smallest prediction
error
      khat<-which.min(colSums((y.test-X.test%*%B)^2))
      theta.hat<-rep(0, ncol(B))
      theta.hat[khat] <- 1
      beta=B[,khat]
      beta.ew=NULL
    }else{#Q-aggregation
      theta.hat<- exp(-colSums((y.test-X.test%*%B)^2)/2)
      theta.hat=theta.hat/sum(theta.hat)
      theta.old=theta.hat
      beta<-as.numeric(B%*%theta.hat)
      beta.ew<-beta
     # theta.old=theta.hat
      for(ss in 1:total.step){
        theta.hat<- exp(-colSums((y.test-X.test%*%B)^2)/2+
colSums((as.vector(X.test%*%beta)-X.test%*%B)^2)/8)
        theta.hat<-theta.hat/sum(theta.hat)
        beta<- as.numeric(B%*%theta.hat*1/4+3/4*beta)
        if(sum(abs(theta.hat-theta.old))<10^(-3)){break}
        theta.old=theta.hat
      }
    }
    list(theta=theta.hat, beta=beta, beta.ew=beta.ew)
  }


  ###oracle Trans-Lasso
  las.kA<-function(X, y, A0, n.vec, lam.const=NULL, l1=T){
    p<-ncol(X)
    size.A0<- length(A0)
    if(size.A0 > 0){
      ind.kA<- ind.set(n.vec, c(1, A0+1))
      ind.1<-1:n.vec[1]
      if(l1){
        y.A<-y[ind.kA]
      }else{ #the l0-method
        y.A<- y[ind.1]
        Sig.hat<-t(X)%*%X/nrow(X)
        for(k in 1:size.A0){
          ind.k<- ind.set(n.vec,k+1)
          lam.k <- sqrt(mean(y[ind.1]^2)/n.vec[1]+mean(y[ind.k]^2)/n.vec[k]) * sqrt(2*log(p))
          delta.hat.k<-lassoshooting(XtX=Sig.hat,
          Xty=t(X[ind.k,])%*%y[ind.k]/n.vec[k+1]-t(X[1:n.vec[1],])%*%y[1:n.vec[1]]/n.vec[1],
          lambda=lam.k)$coef
          y.A<-c(y.A, y[ind.k]-X[ind.k,]%*%delta.hat.k)
        }
      }
      if(is.null(lam.const)){
        cv.init<-cv.glmnet(X[ind.kA,], y.A, nfolds=8, lambda=seq(1,0.1,length.out=10)*sqrt(2*log(p)/length(ind.kA)))
        lam.const <- cv.init$lambda.min/sqrt(2*log(p)/length(ind.kA))
      }
      w.kA <- as.numeric(glmnet(X[ind.kA,], y.A, lambda=lam.const*sqrt(2*log(p)/length(ind.kA)))$beta)
      w.kA<-w.kA*(abs(w.kA)>=lam.const*sqrt(2*log(p)/length(
ind.kA)))
    # cv.delta<-cv.glmnet(x=X[ind.1,],y=y[ind.1]-X[ind.1,]%*%w.kA, lambda=seq(1,0.1,length.out=10)*sqrt(2*log(p)/length(ind.1)))
    #delta.kA<-predict(cv.delta, s='lambda.min', type='coefficients')[-1]
     delta.kA <- as.numeric(glmnet(x=X[ind.1,],y=y[ind.1]-X[ind.1,]%*%w.kA, lambda=lam.const*sqrt(2*log(p)/length(ind.1)))$beta)
     delta.kA<-delta.kA*(abs(delta.kA)>=lam.const*sqrt(2*log(p)/length(ind.1)))
     beta.kA <- w.kA + delta.kA
     lam.const=NA
   }else{
     cv.init<-cv.glmnet(X[1:n.vec[1],], y[1:n.vec[1]], nfolds=8, lambda=seq(1,0.1,length.out=20)*sqrt(2*log(p)/n.vec[1]))
     lam.const<-cv.init$lambda.min/sqrt(2*log(p)/n.vec[1])
     beta.kA <- predict(cv.init, s='lambda.min', type='coefficients')[-1]
     w.kA<-NA
   }
   list(beta.kA=as.numeric(beta.kA),w.kA=w.kA, lam.const=lam.const)

 }

 #Trans Lasso method
 Trans.lasso <- function(X, y, n.vec, I.til, l1=T){
   M= length(n.vec)-1
   #step 1
   X0.til<-X[I.til,] #used for aggregation
   y0.til<-y[I.til]
   X<- X[-I.til,]
   y<-y[-I.til]
   #step 2
   Rhat <- rep(0, M+1)
   p<- ncol(X)
   n.vec[1]<- n.vec[1]-length(I.til)
```

```r
        ind.1<-ind.set(n.vec,1)
        for(k in 2: (M+1)){
          ind.k<-ind.set(n.vec,k)
          Xty.k <- t(X[ind.k,])%*%y[ind.k]/n.vec[k] - t(X[ind.1,])
    %*%y[ind.1]/ n.vec[1]
          margin.T<-sort(abs(Xty.k),decreasing=T)[1:round(n.vec[1]
    /3)]
          Rhat[k] <-  sum(margin.T^2)
        }
        Tset<- list()
        k0=0
        kk.list<-unique(rank(Rhat[-1]))
        #cat(rank(Rhat[-1]),'\n')
        for(kk in 1:length(kk.list)){#use Rhat as the selection
    rule
          Tset[[k0+kk]]<- which(rank(Rhat[-1]) <= kk.list[kk])
        }
        k0=length(Tset)
        Tset<- unique(Tset)
        #cat(length(Tset),'\n')

        beta.T<-list()
        init.re<-las.kA(X=X, y=y, A0=NULL, n.vec=n.vec, l1=l1)
        beta.T[[1]] <- init.re$beta.kA
        beta.pool.T<-beta.T ##another method for comparison
        for(kk in 1:length(Tset)){#use pi.hat as selection rule
          T.k <- Tset[[kk]]
          re.k<- las.kA(X=X, y=y, A0=T.k, n.vec=n.vec, l1=l1, lam.
    const=init.re$lam.const)
          beta.T[[kk+1]] <-re.k$beta.kA
          beta.pool.T[[kk+1]]<-re.k$w.kA
        }
        beta.T<-beta.T[!duplicated((beta.T))]
        beta.T<- as.matrix(as.data.frame(beta.T))
        agg.re1 <- agg.fun(B=beta.T, X.test=X0.til, y.test=y0.til)
        beta.pool.T<-beta.pool.T[!duplicated((beta.pool.T))]
        beta.pool.T<- as.matrix(as.data.frame(beta.pool.T))
        agg.re2<-agg.fun(B=beta.pool.T, X.test=X0.til, y.test=y0.
    til)

        return(list(beta.hat=agg.re1$beta, theta.hat=agg.re1$theta, rank.pi=rank(Rhat[-1]),
        beta.pool=agg.re2$beta, theta.pool=agg.re2$theta))
      }


      #A method for comparison: Trans-Lasso(l1). It has the same pipeline of the Trans-Lasso
      ###but with sparsity index R_k=\|w^{(k)}-\beta\|_1 and a naive aggregation (empirical risk minimization)
      Trans.lasso.sp <- function(X, y, n.vec, I.til, l1=T){
        M= length(n.vec)-1
        #step 1
        X0.til<-X[I.til,] #used for aggregation
        y0.til<-y[I.til]
        X<- X[-I.til,]
        y<-y[-I.til]
        #step 2
        Rhat <- rep(0, M+1)
        p<- ncol(X)
        n.vec[1]<- n.vec[1]-length(I.til)
        ind.1<-ind.set(n.vec,1)
        init.re<-las.kA(X=X, y=y, A0=NULL, n.vec=n.vec, l1=l1)
        for(k in 2: (M+1)){
          ind.k <- ind.set(n.vec,k)
          w.init.k<- as.numeric(glmnet(X[ind.k,], y[ind.k], lambda=init.re$lam.const*sqrt(2*log(p)/length(ind.k)))$beta)
          Rhat[k] <-  sum(abs(w.init.k-init.re$beta.kA)) ##\|w^{(k)}-\beta\|_1
        }
        Tset<- list()
        k0=0
        kk.list<-unique(rank(Rhat[-1]))
        #cat(rank(Rhat[-1]),'\n')
        for(kk in 1:length(kk.list)){#use pi.hat as selection rule
          Tset[[k0+kk]]<- which(rank(Rhat[-1]) <= kk.list[kk])
        }
        k0=length(Tset)
        Tset<- unique(Tset)
        # cat(length(Tset),'\n')

        beta.T<-list()
        beta.T[[1]] <- init.re$beta.kA
        for(kk in 1:length(Tset)){#use pi.hat as selection rule
          T.k <- Tset[[kk]]
          beta.T[[kk+1]] <-las.kA(X=X, y=y, A0=T.k, lam.const=init
    .re$lam.const,n.vec=n.vec, l1=l1)$beta.kA
        }
        beta.T<-beta.T[!duplicated((beta.T))]
        beta.T<- as.matrix(as.data.frame(beta.T))
        agg.re <- agg.fun(B=beta.T, X.test=X0.til, y.test=y0.til,
    selection =T)
        return(list(beta.sp=agg.re$beta, theta.sp=agg.re$theta,
    rank.pi=rank(Rhat[-1])))
      }

      #computing the MSE
      mse.fun<- function(beta,est, X.test=NULL){
        pred.err<-NA
        est.err<- sum((beta-est)^2)

        if(!is.null(X.test)){
          pred.err<-  mean((X.test%*%(beta-est))^2)
        }
        return(list(est.err=est.err, pred.err= pred.err))
      }

      ind.set<- function(n.vec, k.vec){
        ind.re <- NULL
        for(k in k.vec){
          if(k==1){
            ind.re<-c(ind.re,1: n.vec[1])
          }else{
            ind.re<- c(ind.re, (sum(n.vec[1:(k-1)])+1): sum(n.vec
    [1:k]))
          }
        }
        ind.re
      }
      rep.col<-function(x,n){
        matrix(rep(x,each=n), ncol=n, byrow=TRUE)
      }


```

Code for plot

```
        library(tidyverse)


        Coef.gen<- function(s, h,q=30, size.A0, M, sig.beta,sig.
    delta1, sig.delta2, p, exact=T){
          beta0 <- c(rep(sig.beta,s), rep(0, p - s))
         W <- rep.col(beta0,  M)# ten prior estimates
         W[1,]<-W[1,]-2*sig.beta
          for(k in 1:M){
            if(k <= size.A0){
              if(exact){
                samp0<- sample(1:p, h, replace=F)
                W[samp0,k] <-W[samp0,k] + rep(-sig.delta1, h)
              }else{
                W[1:100,k] <-W[1:100,k] + rnorm(100, 0, h/100)
              }
            }else{
              if(exact){
                samp1 <- sample(1:p, q, replace = F)
                W[samp1,k] <- W[samp1,k] + rep(-sig.delta2,q)
              }else{
                W[1:100,k] <-W[1:100,k] + rnorm(100, 0, q/100)
              }
            }
          }
          return(list(W=W, beta0=beta0))
        }


        fig1 <- function(h,size.A0, exact = F){
          p = 500
          s = 16
         M = 20
          sig.beta = 0.3

          sig.z <- 1
          n0 <- 150
         M = 20
          n.vec <- c(n0, rep(100, M))
          Sig.X <- diag(1, p)
          Niter = 500
          l1=T

          A0 = 1:size.A0
          beta0<-
          coef.all <-Coef.gen( s, h = h, q = 2*s, size.A0 = size.A0,
       M = M,    sig.beta = sig.beta,
          sig.delta1 = sig.beta, sig.delta2 = sig.beta+0.2, p = p,
    exact=exact)
          B <- cbind(coef.all$beta0, coef.all$W)
          beta0 <- coef.all$beta0
          library(mvtnorm)
         ###generate the data
         X <- NULL
          y <- NULL
          for (k in 1:(M + 1)) {
            X <- rbind(X, rmvnorm(n.vec[k], rep(0, p), Sig.X))
            ind.k <- ind.set(n.vec, k)
            y <- c(y, X[ind.k, ] %*% B[, k] + rnorm (n.vec[k], 0, 1)
    )
          }
         ###compute init beta ####
```

```r
    mse.vec<-rep(NA,6)
    beta.init <-
    as.numeric(glmnet(X[1:n.vec[1], ], y[1:n.vec[1]], lambda = sqrt(2 * log(p) / n.vec[1]))$beta)
    mse.vec[1] = mse.fun(as.numeric(beta.init), beta0)$est.err

    ######Oracle Trans-Lasso#######
    if (size.A0 == 0) {
      beta.kA <- beta.init
    } else{
      beta.kA <- las.kA(X, y, A0 = 1:size.A0, n.vec = n.vec, l1=l1)$beta.kA
    }
    mse.vec[2] = mse.fun(as.numeric(beta.kA), beta0)$est.err
    ###########Trans-Lasso##############
    prop.re1 <- Trans.lasso(X, y, n.vec, I.til = 1:50, l1 = l1)
    prop.re2 <- Trans.lasso(X, y, n.vec, I.til = 101:n.vec[1], l1=l1)
    if(size.A0 > 0 & size.A0< M){ #Rank.re characterizes the performance of the sparsity index Rk
      Rank.re<- (sum(prop.re1$rank.pi[1:size.A0]<=size.A0) +
      sum(prop.re2$rank.pi[1:size.A0]<=size.A0))/2/size.A0
    }else{ Rank.re <- 1 }
    beta.prop <- (prop.re1$beta.hat + prop.re2$beta.hat) / 2
    mse.vec[3] = mse.fun(beta.prop, beta0)$est.err

    ######A method for comparison: it has the same pipeline of the Trans-Lasso
    ###but with sparsity index R_k=\|w^{(k)}-\beta\|_1 and a naive aggregation (empirical risk minimization)
    prop.sp.re1 <- Trans.lasso.sp(X, y, n.vec, I.til = 1:50, l1 = l1)
    prop.sp.re2 <- Trans.lasso.sp(X, y, n.vec, I.til = 101:n.vec[1], l1=l1)
    if(size.A0 > 0 & size.A0< M){
      Rank.re.sp <- (sum(prop.sp.re1$rank.pi[1:size.A0]<=size.A0) +
      sum(prop.sp.re2$rank.pi[1:size.A0]<=size.A0))/2/size.A0
    }else{ Rank.re.sp <-1 }
    beta.sp <- (prop.sp.re1$beta.sp + prop.sp.re2$beta.sp) / 2
    mse.vec[4] = mse.fun(beta.sp, beta0)$est.err

    ######another method for comparison: it is the same as Trans-Lasso except
    ##that the bias correction step (step 2 of Oracle Trans-Lasso) is omitted
    beta.pool<-(prop.re1$beta.pool+prop.re2$beta.pool)/2
    mse.vec[5] = mse.fun(beta.pool, beta0)$est.err
    #####naive translasso: simply assumes A0=1:K
    beta.all <- las.kA(X, y, A0 = 1:M, n.vec = n.vec, l1=l1)$beta.kA#naive transLasso
    mse.vec[6] = mse.fun(as.numeric(beta.all), beta0)$est.err
    return(mse.vec)
  }

  h = c(2,6,12)
  size.A0 = c(0,4,8,12,16,20)

  plot.sse = function(data,i){
    data %>% gather(key = "method", value = "SSE", -A) %>%
    ggplot(aes(x = A, y = SSE, group = method, color = method, shape = method)) +
    geom_point()+
    geom_line() +
    ylab(paste0("SSE (h=",i,")")) +
    theme_bw()
  }

  plot.data <- function(sse,i){
    data = sse[[i]][,-6]
    data = as.data.frame(data)
    names(data) = c("Lasso","Oracle-Trans-Lasso","Trans-Lasso","Agg-Lasso","hc-l1")
    data$"A" = size.A0
    plot.sse(data, h[i])
```

```
  }

  sse11 = list()

  for (i in 1:3){
    ssevec = matrix(NA, ncol = 6, nrow = 6)
    for(j in 1:6){
      ssevec[j, ] = fig1(h[i], size.A0[j])
    }
    sse11[[i]] = ssevec
  }

  library(ggpubr)

  p11.1  = plot.data(sse11, 1)
  p11.2  = plot.data(sse11, 2)
  p11.3  = plot.data(sse11, 3)

  ggarrange(p11.1, p11.2, p11.3, ncol=3, nrow=1, common.legend
= TRUE, legend="bottom")

  sse12 = list()
  for (i in 1:3){
    ssevec = matrix(NA, ncol = 6, nrow = 6)
    for(j in 1:6){
      ssevec[j, ] = fig1(h[i], size.A0[j], exact = T)
    }
    sse12[[i]] = ssevec
  }

  p12.1  = plot.data(sse12, 1)
  p12.2  = plot.data(sse12, 2)
  p12.3  = plot.data(sse12, 3)

  ggarrange(p12.1, p12.2, p12.3, ncol=3, nrow=1, common.legend
= TRUE, legend="bottom")

  fig2 <- function(h,size.A0, exact = F){
        p = 500
        s = 16
        M = 20
        sig.beta = 0.3

        sig.z <- 1
        n0 <- 150
        M = 20
        n.vec <- c(n0, rep(100, M))
        Sig.X <- diag(1, p)
        Niter = 200
        l1=T

        A0 = 1:size.A0
        beta0<-
        coef.all <-Coef.gen( s, h = h, q = 2*s, size.A0 = size.A0,
   M = M,     sig.beta = sig.beta,
        sig.delta1 = sig.beta, sig.delta2 = sig.beta+0.2, p = p,
exact=exact)
        B <- cbind(coef.all$beta0, coef.all$W)
        beta0 <- coef.all$beta0
        library(mvtnorm)
        ###generate the data
        X <- NULL
        y <- NULL
        for (k in 1:(M + 1)) {
          if(k<=size.A0+1)
          {Sig.X[which(Sig.X == 0)] = 0.8}
          else{
            Sig.X[1, (1+1):(2*k-1+1)] = 1/(k+1)
            Sig.X[(1+1):(2*k-1+1),1] = 1/(k+1)}
          X <- rbind(X, rmvnorm(n.vec[k], rep(0, p), Sig.X))
          ind.k <- ind.set(n.vec, k)
          y <- c(y, X[ind.k, ] %*% B[, k] + rnorm (n.vec[k], 0, 1)
)
        }
        ###compute init beta ####
        mse.vec<-rep(NA,6)
```

```
    beta.init <-
    as.numeric(glmnet(X[1:n.vec[1], ], y[1:n.vec[1]], lambda = sqrt(2 * log(p) / n.vec[1]))$beta)
    mse.vec[1] = mse.fun(as.numeric(beta.init), beta0)$est.err

    ######Oracle Trans-Lasso#######
    if (size.A0 == 0) {
      beta.kA <- beta.init
    } else{
      beta.kA <- las.kA(X, y, A0 = 1:size.A0, n.vec = n.vec, l1=l1)$beta.kA
    }
    mse.vec[2] = mse.fun(as.numeric(beta.kA), beta0)$est.err
    ###########Trans-Lasso#############
    prop.re1 <- Trans.lasso(X, y, n.vec, I.til = 1:50, l1 = l1)
    prop.re2 <- Trans.lasso(X, y, n.vec, I.til = 101:n.vec[1], l1=l1)
    if(size.A0 > 0 & size.A0< M){ #Rank.re characterizes the performance of the sparsity index Rk
      Rank.re<- (sum(prop.re1$rank.pi[1:size.A0]<=size.A0) +
      sum(prop.re2$rank.pi[1:size.A0]<=size.A0))/2/size.A0
    }else{ Rank.re <- 1 }
    beta.prop <- (prop.re1$beta.hat + prop.re2$beta.hat) / 2
    mse.vec[3] = mse.fun(beta.prop, beta0)$est.err

    ######A method for comparison: it has the same pipeline of the Trans-Lasso
    ###but with sparsity index R_k=\|w^{(k)}-\beta\|_1 and a naive aggregation (empirical risk minimization)
    prop.sp.re1 <- Trans.lasso.sp(X, y, n.vec, I.til = 1:50, l1 = l1)
    prop.sp.re2 <- Trans.lasso.sp(X, y, n.vec, I.til = 101:n.vec[1], l1=l1)
    if(size.A0 > 0 & size.A0< M){
      Rank.re.sp <- (sum(prop.sp.re1$rank.pi[1:size.A0]<=size.A0) +
      sum(prop.sp.re2$rank.pi[1:size.A0]<=size.A0))/2/size.A0
    }else{ Rank.re.sp <-1 }
    beta.sp <- (prop.sp.re1$beta.sp + prop.sp.re2$beta.sp) / 2
    mse.vec[4] = mse.fun(beta.sp, beta0)$est.err

    ######another method for comparison: it is the same as Trans-Lasso except
    ##that the bias correction step (step 2 of Oracle Trans-Lasso) is omitted
    beta.pool<-(prop.re1$beta.pool+prop.re2$beta.pool)/2
    mse.vec[5] = mse.fun(beta.pool, beta0)$est.err
    #####naive translasso: simply assumes A0=1:K
    beta.all <- las.kA(X, y, A0 = 1:M, n.vec = n.vec, l1=l1)$beta.kA#naive transLasso
    mse.vec[6] = mse.fun(as.numeric(beta.all), beta0)$est.err
    return(mse.vec)
  }

  sse21 = list()

  for (i in 1:3){
    ssevec = matrix(NA, ncol = 6, nrow = 6)
    for(j in 1:6){
      ssevec[j, ] = fig2(h[i], size.A0[j])
    }
    sse21[[i]] = ssevec
  }

  p21.1  = plot.data(sse21, 1)
  p21.2  = plot.data(sse21, 2)
  p21.3  = plot.data(sse21, 3)
  ggarrange(p21.1, p21.2, p21.3, ncol=3, nrow=1, common.legend = TRUE, legend="bottom")

  sse22 = list()
  for (i in 1:3){
    ssevec = matrix(NA, ncol = 6, nrow = 6)
    for(j in 1:6){
      ssevec[j, ] = fig2(h[i], size.A0[j], exact = T)
    }
    sse22[[i]] = ssevec
  }

  p22.1  = plot.data(sse22, 1)
  p22.2  = plot.data(sse22, 2)
  p22.3  = plot.data(sse22, 3)
  ggarrange(p22.1, p22.2, p22.3, ncol=3, nrow=1, common.legend
 = TRUE, legend="bottom")

  fig3 <- function(h,size.A0, exact = F){
    p = 500
    s = 16
    M = 20
    sig.beta = 0.3

    sig.z <- 1
    n0 <- 150
    M = 20
    n.vec <- c(n0, rep(100, M))
    Sig.X <- diag(1, p)
    Niter = 200
    l1=T

    A0 = 1:size.A0
    beta0<-
    coef.all <-Coef.gen( s, h = h, q = 2*s, size.A0 = size.A0,
  M = M,   sig.beta = sig.beta,
    sig.delta1 = sig.beta, sig.delta2 = sig.beta+0.2, p = p,
exact=exact)
    B <- cbind(coef.all$beta0, coef.all$W)
    beta0 <- coef.all$beta0
    library(mvtnorm)
    ###generate the data
    X <- NULL
    y <- NULL
    X <- rbind(X, rmvnorm(n.vec[1], rep(0, p), Sig.X))
    ind.1 <- ind.set(n.vec, 1)
```

```r
    y <- c(y, X[ind.1, ] %*% B[, 1] + rnorm (n.vec[1], 0, 1))
    for (k in 2:(M + 1)) {
      Sig.X[1, (1+1):(2*k-1+1)] = 1/(k+1)
      Sig.X[(1+1):(2*k-1+1),1] = 1/(k+1)
      X <- rbind(X, rmvnorm(n.vec[k], rep(0, p), Sig.X))
      ind.k <- ind.set(n.vec, k)
      y <- c(y, X[ind.k, ] %*% B[, k] + rnorm (n.vec[k], 0, 1)
)
    }
    ###compute init beta ####
    mse.vec<-rep(NA,6)
    beta.init <-
    as.numeric(glmnet(X[1:n.vec[1], ], y[1:n.vec[1]], lambda =
 sqrt(2 * log(p) / n.vec[1]))$beta)
    mse.vec[1] = mse.fun(as.numeric(beta.init), beta0)$est.err

    ######Oracle Trans-Lasso#######
    if (size.A0 == 0) {
      beta.kA <- beta.init
    } else{
      beta.kA <- las.kA(X, y, A0 = 1:size.A0, n.vec = n.vec,
l1=l1)$beta.kA
    }
    mse.vec[2] = mse.fun(as.numeric(beta.kA), beta0)$est.err
    ###########Trans-Lasso#############
    prop.re1 <- Trans.lasso(X, y, n.vec, I.til = 1:50, l1 = l1
)
    prop.re2 <- Trans.lasso(X, y, n.vec, I.til = 101:n.vec[1],
 l1=l1)
    if(size.A0 > 0 & size.A0< M){ #Rank.re characterizes the
performance of the sparsity index Rk
      Rank.re<- (sum(prop.re1$rank.pi[1:size.A0]<=size.A0) +
      sum(prop.re2$rank.pi[1:size.A0]<=size.A0))/2/size.A0
    }else{ Rank.re <- 1 }
    beta.prop <- (prop.re1$beta.hat + prop.re2$beta.hat) / 2
    mse.vec[3] = mse.fun(beta.prop, beta0)$est.err

    ######A method for comparison: it has the same pipeline of
 the Trans-Lasso
    ###but with sparsity index R_k=\|w^{(k)}-\beta\|_1 and a naive aggregation (empirical risk minimization)
    prop.sp.re1 <- Trans.lasso.sp(X, y, n.vec, I.til = 1:50, l1 = l1)
    prop.sp.re2 <- Trans.lasso.sp(X, y, n.vec, I.til = 101:n.vec[1], l1=l1)
    if(size.A0 > 0 & size.A0< M){
      Rank.re.sp <- (sum(prop.sp.re1$rank.pi[1:size.A0]<=size.A0) +
      sum(prop.sp.re2$rank.pi[1:size.A0]<=size.A0))/2/size.A0
    }else{ Rank.re.sp <-1 }
    beta.sp <- (prop.sp.re1$beta.sp + prop.sp.re2$beta.sp) / 2
    mse.vec[4] = mse.fun(beta.sp, beta0)$est.err

    ######another method for comparison: it is the same as Trans-Lasso except
    ##that the bias correction step (step 2 of Oracle Trans-Lasso) is omitted
    beta.pool<-(prop.re1$beta.pool+prop.re2$beta.pool)/2
    mse.vec[5] = mse.fun(beta.pool, beta0)$est.err
    #####naive translasso: simply assumes A0=1:K
    beta.all <- las.kA(X, y, A0 = 1:M, n.vec = n.vec, l1=l1)$beta.kA#naive transLasso
    mse.vec[6] = mse.fun(as.numeric(beta.all), beta0)$est.err
    return(mse.vec)
  }

  sse31 = list()

  for (i in 1:3){
    ssevec = matrix(NA, ncol = 6, nrow = 6)
    for(j in 1:6){
      ssevec[j, ] = fig3(h[i], size.A0[j])
    }
    sse31[[i]] = ssevec
  }

```

```
    p31.1  = plot.data(sse31, 1)
    p31.2  = plot.data(sse31, 2)
    p31.3  = plot.data(sse31, 3)
    ggarrange(p31.1, p31.2, p31.3, ncol=3, nrow=1, common.legend
  = TRUE, legend="bottom")

    sse32 = list()
    for (i in 1:3){
      ssevec = matrix(NA, ncol = 6, nrow = 6)
      for(j in 1:6){
        ssevec[j, ] = fig3(h[i], size.A0[j], exact = T)
      }
      sse32[[i]] = ssevec
    }

    p32.1  = plot.data(sse32, 1)
    p32.2  = plot.data(sse32, 2)
    p32.3  = plot.data(sse32, 3)
    ggarrange(p32.1, p32.2, p32.3, ncol=3, nrow=1, common.legend
  = TRUE, legend="bottom")

```