Mpersonalized Vignette

Chensheng Kuang

March 29, 2018

Contents

1 Introduction

`mpersonalized` is an R package aiming to solve the problem of personalized medicine in meta-analysis and multiple outcomes. This package extends the contrast classification framework (B. Zhang et al. 2012) to both meta-analysis and multiple outcomes, and fits a linear classifier to identify the subgroup who might benefit from treatment. With an increasing number of biomarkers available nowadays, variable selection has become more than necessary in personalized medicine. Hence, the package incorporate a rich variety of variable selection methods to handle high dimensional problems properly. `mpersonalized` also provides other options flexible enough to meet different requirements from users; e.g. in multiple outcomes, using different rules or a single rule.

In the setting of a single study or outcome, the contrast classification framework (B. Zhang et al. 2012) estimates the optimal treatment rule by solving

$$\min_{g} \sum_{i=1}^{n} |\hat{C}(X_i)|[1\{\hat{C}(X_i) > 0\} - g(X_i)]^2, \tag{1}$$

where $\hat{C}$ is the estimator of the usually unknown contrast function $C(X) = E(Y|A = 1, X) - E(Y|A = 0, X)$ and $g(X)$ is a treatment recommendation rule. Then, given a subject with baseline covariates x, $A = 1$ is recommended if $g(x) > 0.5$ and vice versa. Althgouh adapting the framework to more complicated settings, `mpersonalized` incorporates the whole pipeline, from estimation of the contrast function to fitting of treatment rules to eventually prediction for new subjects, in one single package so as to facilitate the usage of this framework.

The main functions in our package are `mpersonalized` and `mpersonalized_cv` and this vignette will focus on the usage of the main functions. Various arguments can be passed to these functions to best meet user requirements. We divide the optional arguments into different groups and place them in different sections of the vignette to ease understanding. The vignette is structured as follows: in section 2, we briefly describe how to install `mpersonalized` package; in section 3, we give a quick example to demonstrate the typical usage; in section 4 and section 5, we introduce the arguments related to data input and model specification, respectively; in section 6, different ways of sepcifying penalty parameter sequence are introduced; finally, in section 7, we provide users with the avaiable S3 methods for return values from `mpersonalized` and `mpersonalized_cv`. There are also some more sophisticated arguments, which do not usually requires specification from users, and thus they are covered in Appendix.

This vignette is aimed to help users understand how `mpersonalized` works and how to apply it to a new dataset or problem according to specific requirements. Various examples will be provided and most options will be covered. To learn more about the package, we recommended users to the reference manual.

2 Installation

`mpersonalized` can be installed from Github repository "chenshengkuang/mpersonalized" now. In order to do that, users need to install R package `devtools` first. `devtools` could be downloaded from CRAN by the following command:

```
install.packages("devtools")
```

When `devtools` has been installed, we could then install `mpersonalized` package using commands:

```
library(devtools)
install_github("chenshengkuang/mpersonalized")
```

In the future, there might be frequent updates of the package. To update `mpersonalized`, install it again using the above commands.

3 A Quick Example

In this section, we provide a quick example to illustrate the typical usage and functionality of `mpersonalized` package. This example is demonstrated through a simulated dataset generated by the built-in function `simulated_dataset` in this package.

We first load the package and generate a simulated meta-analysis dataset.

```
# Load package
library(mpersonalized)

set.seed(123)
```

```r
# Generate simulated dataset
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")
Xlist = sim_dat$Xlist; Ylist = sim_dat$Ylist; Trtlist = sim_dat$Trtlist
```

The simulated dataset contains the information of 6 studies, with each of them containing information of 200 subjects. Specifically, `Xlist` contains all the information of baseline covariates, `Ylist` all the information of outcome and `Trtlist` all the information of treatment indicator. To inspect the structure of the simulated dataset, users can apply R base function `str` to `Xlist`, `Ylist` and `Trtlist`.

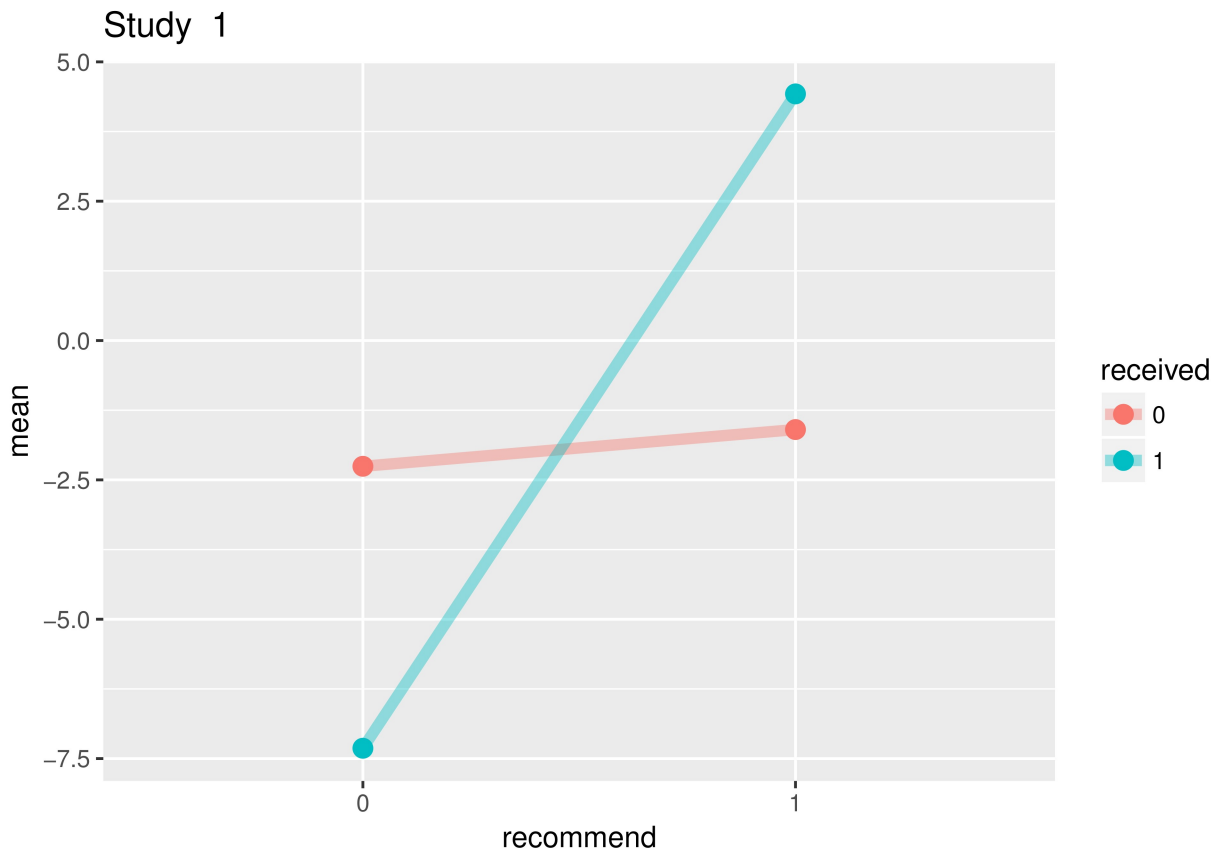
The simulated dataset has 50 baseline covariates with only 200 subjects in each of the studies. Therefore, regularization is necessary not only to reduce the variance but also for variable selection so that the fitted model is more interpretable.

Using function `mpersonalized_cv`, we could fit different rules for different stuides and perform variable selection with a penalty function. As we mentioned in introduction, `mpersonalized_cv` and `mpersonalized` are the main functions of the package. The difference between them is that `mpersonalized` fit a model for each different penalty parameter in the penalty parameter sequence while `mpersonalized_cv` will further tune the penalty parameter by built-in cross validation and provide the optimal model.

```r
mp_mod = mpersonalized_cv(problem = "meta-analysis",
                          Xlist = Xlist, Ylist = Ylist, Trtlist = Trtlist,
                          penalty = "lasso", single_rule = FALSE)
```

`mp_mod`, the return value from `mpersonalized_cv`, could then be passed to other functions in the `mpersonalized` package for analysis. For example, `plot(mp_mod)` will return a list object containing the interaction plots of the fitted rules. These plots could be used as a sanity check to see whether the rules are working as intended, at least on training set.

```r
interaction_plots = plot(mp_mod)

# Interaction plot for the first study
interaction_plots[[1]]
```



We could also predict optimal treatments for new subejcts using `predict` function.

```
# Simulate baseline covariates for 10 new subjects
newx = matrix(rnorm(10 * 50), nrow = 10)

pred = predict(mp_mod, newx = newx)
pred$opt_treatment
```

```
##       Study 1 Study 2 Study 3 Study 4 Study 5 Study 6 Overall Rec
##  [1,]       0       0       0       0       0       0            0
##  [2,]       0       0       0       0       0       0            0
##  [3,]       0       0       0       0       0       0            0
##  [4,]       0       0       0       0       0       0            0
##  [5,]       1       1       0       0       0       0            0
##  [6,]       0       1       0       0       0       0            0
##  [7,]       0       0       0       0       0       0            0
##  [8,]       0       0       0       0       0       0            0
##  [9,]       0       0       1       1       1       1            1
## [10,]       1       1       1       1       1       0            1
```

```
set.seed(NULL)
```

`pred$opt_treatment` displays the optimal treatments based on study treatment rules. It also gives an additional column named as overall treatment recommendation, which can be viewed as a weighted voting of recommendations from each study and the weight can be specified by users. Other than prediction for new subejcts, `predict` function can also be convenient for validation purpose if users have a validation set.

This quick example is merely aimed to provide a general understanding of what `mpersonalized` is capable of

doing. In the rest of the vignette, we will explain more carefully about the functions and step by step how users can set theses options to best fit their problems.

4 Data Arguments

Before feeding their data to `mpersonalized` or `mpersonalized_cv`, the very first thing for users to do is to specify whether they are solving a "meta-analysis" problem or a "multiple outcomes" problem. This is denoted by the argument `problem` in function `mpersonalized` and `mpersonalized_cv`. Users should always specify `problem` so that these functions can choose the correct set of arguments.

`problem = "meta-analysis"` refers to the case when multiple studies study the effect of the same treatment, and in our setting, it is further assumed the studies are using the same outcome to evaluate efficacy and the same set of baseline covariates as well. The subjects from study to study could be potentially different. In other words, the baseline covariates matrices could be different.

`problem = "multiple outcomes"`, on the other hand, refers to the case when there is only one study but multiple outcomes are observed for every subject. Outcomes can be different types of variables, for example, one is binary and another is continuous. There is, however, only one single baseline covariates matrix because these outcomes are observed from the same subjects.

4.1 Data Input Formats

The above difference results in different data input formats. When `problem = "meta-analysis"`, `mpersonalized` and `mpersonalized_cv` require a baseline covariates matrix, a treatment vector and an outcome vector from each study; when `problem = "multiple outcomes"`, they instead require a single baseline covariates matrix, a single treatment vector and multiple outcome vectors.

We illustrate the difference through function `simulated_dataset`. The simulated dataset is formulated so that it already satisfies the input formats of `mpersonalized` and `mpersonalized_cv`.

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat1 = simulated_dataset(n = 200, problem = "meta-analysis")
str(sim_dat1, vec.len = 1)
```

```
## List of 4
##  $ Xlist  :List of 6
##   ..$ : num [1:200, 1:50] 0.235 ...
##   ..$ : num [1:200, 1:50] -0.211 ...
##   ..$ : num [1:200, 1:50] 0.521 ...
##   ..$ : num [1:200, 1:50] -0.24 ...
##   ..$ : num [1:200, 1:50] -0.162 ...
##   ..$ : num [1:200, 1:50] 1.05 ...
##  $ Ylist  :List of 6
##   ..$ : num [1:200, 1] -2.24 ...
##   ..$ : num [1:200, 1] -10 ...
##   ..$ : num [1:200, 1] -11.3 ...
##   ..$ : num [1:200, 1] 3.59 ...
##   ..$ : num [1:200, 1] 4.22 ...
##   ..$ : num [1:200, 1] -5.97 ...
##  $ Trtlist:List of 6
##   ..$ : int [1:200] 1 1 ...
##   ..$ : int [1:200] 1 1 ...
##   ..$ : int [1:200] 0 1 ...
```

```
##   ..$ : int [1:200] 1 0 ...
##   ..$ : int [1:200] 0 0 ...
##   ..$ : int [1:200] 0 0 ...
##  $ B      : num [1:6, 1:102] -1.8 -1.8 ...
```

```
# Simulated dataset for multiple outcomes
sim_dat2 = simulated_dataset(n = 200, problem = "multiple outcomes")
str(sim_dat2, vec.len = 1)
```

```
## List of 4
##  $ X    : num [1:200, 1:50] 1.11 ...
##  $ Ylist:List of 6
##   ..$ : num [1:200, 1] -4.61 ...
##   ..$ : num [1:200, 1] -4.65 ...
##   ..$ : num [1:200, 1] -6.74 ...
##   ..$ : num [1:200, 1] -5.25 ...
##   ..$ : num [1:200, 1] -4.57 ...
##   ..$ : num [1:200, 1] -5.25 ...
##  $ Trt  : int [1:200] 1 0 ...
##  $ B    : num [1:6, 1:102] -1.8 -1.8 ...
```

When `problem = "meta-analysis"`, `Xlist`, `Ylist` and `Trtlist` should be provided to function `mpersonalized` and `mpersonalized_cv`. `Xlist`, `Ylist` and `Trtlist` should all be lists, with each element of these lists representing the baseline covariates matrix, the outcome vector and the treatment vector of each study, respectively. When `problem = "multiple outcomes"`, users provide `X`, `Ylist` and `Trt` instead, where `X` and `Trt` denote the baseline covariates matrix and treatment vector, and `Ylist` is a list with each element denoting one of the multiple outcomes.

We can fit a model for each of the above simulated dataset in the following fashion.

```
# Fit for meta-analysis dataset
mp_mod1 = mpersonalized_cv(problem = "meta-analysis",
                           Xlist = sim_dat1$Xlist,
                           Ylist = sim_dat1$Ylist,
                           Trtlist = sim_dat1$Trtlist,
                           penalty = "lasso", single_rule = FALSE)

# Fit for multiple outcomes dataset
mp_mod2 = mpersonalized_cv(problem = "multiple outcomes",
                           X = sim_dat2$X,
                           Ylist = sim_dat2$Ylist,
                           Trt = sim_dat2$Trt,
                           penalty = "lasso", single_rule = FALSE)
set.seed(NULL)
```

4.2 Other Data Arguments

Propensity score can be supplied via arguments `Plist` when `problem = "meta-analysis"` or `P` when `problem = "multiple outcomes"`. The difference is `Plist` is a list object containing the propensity score vectors for subjects in multiple studies, while `P` is the propensity score vector for the one and only group of subjects. Although propensity score is not mandatory for `mpersonalized` and `mpersonalized_cv`, we do recommend users to specify it especially when dealing with observational data. If not supplied, the data will be treated as randomized experiments and the propensity score will be esimated as the proportion of treated units for every individual.

Another argument related to data is `typelist`. This argument specifies the type of the outcome for each

element in `Ylist`, and is helpful in estimation of the contrast function better. Currently each element in `typelist` could only take value from `"continuous"`or `"binary"` and default value is `"continuous"`.

To summarize, depending on the argument `problem`, users should choose the correct set of arguments (`Xlist`, `Ylist`, `Trtlist`, `Plist` for `problem = "meta-analysis"` and `X`, `Ylist`, `Trt`, `P` for `problem = "multiple outcomes"`) for input and the input should satisfies corresponding formats.

5 Model Arguments

In terms of modeling aspect, the most critical argument to feed to `mpersonalized` and `mpersonalized_cv` is `single_rule`, a logical value with `FALSE` as default value. This arugment specifies whether a single treatment rule should be fitted across all studies if `problem = "meta-analysis"` and across all outcomes if `problem = "multiple outcomes"`. Letting `single_rule` be `TRUE` or `FALSE` will formulate different optimization problems, and this decision should depend on the context of the problem.

5.1 `single_rule = FALSE`

If each study or outcome has its own treatment recommendation rule, denoted as $g_i(X)$ for ith study or ith outcome, then based on the contrast classification framework 1, the objective function of optimization is formulated as

$$\min_{g_1,\ldots,g_K} \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{n_k} \frac{|\hat{C}_k(X_{ki})|}{\sum_{i=1}^{n_k}|\hat{C}_k(X_{ki})|}\left[1\{\hat{C}_k(X_{ki})>0\} - g_k(X_{ki})\right]^2 + h(g_1,\ldots,g_K) \tag{2}$$

for meta-analysis and

$$\min_{g_1,\ldots,g_K} \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{n} \frac{|\hat{C}_k(X_i)|}{\sum_{i=1}^{n}|\hat{C}_k(X_i)|}\left[1\{\hat{C}_k(X_i)>0\} - g_k(X_i)\right]^2 + h(g_1,\ldots,g_K) \tag{3}$$

for multiple outcomes.

Note in the above formulation, the loss function of each study or outcome in 1 is adjusted by its contrast estimator, to avoid the issue of contrast estimators not being on the same scale. $h(g_1,\ldots,g_K)$, a regularization term, is useful when variable selection is needed. `mpersonalized` package focuses on linear classifiers and we denote $g_j(X) = \beta_{0j} + \beta_{1j}X_1 + \cdots + \beta_{pj}X_j$. h is a combination of sparse group lasso penalty and fused lasso penalty,

$$h = (1-\alpha)\lambda_1\sqrt{q}\sum_{j=1}^{p}\|\boldsymbol{\beta_j}\|_2 + \alpha\lambda_1\sum_{j=1}^{p}\|\boldsymbol{\beta_j}\|_1 + \lambda_2\sum_{j=1}^{p}\sum_{1\le a<b\le K}|\beta_{ja}-\beta_{jb}| \tag{4}$$

where $\boldsymbol{\beta_j} = (\beta_{j1},\ldots,\beta_{jK})$. This formulation provides a flexible choice of penalty when fitting $g_1,\ldots,g_K$. Group lasso penalty is helpful when rules share common variables and fused lasso penalty is helpful when the coefficients are close. Users are free to drop or keep each penalty term depending on the detailed problem.

When using `mpersonalized` and `mpersonalized_cv`, it is recommended users specify the argument `penalty` and then either use default penalty parameter sequence or supply the functions a specified sequence. For `mpersonalized`, the argument `penalty` could be choosen from the following options, covering all the possible combintaion of penalty terms:

- "none": no penalty. $\lambda_1 = 0$, $\lambda_2 = 0$.
- "lasso": lasso penalty. $\alpha = 1$, $\lambda_1 \neq 0$, $\lambda_2 = 0$.

- "GL": group lasso penalty. $\alpha = 0$, λ_1 $ne0$, $\lambda_2 = 0$.
- "SGL": sparse group lasso penalty. $\alpha \neq 0$ or 1, $\lambda_1 \neq 0$, $\lambda_2 = 0$.
- "fused": fused lasso penalty. $\lambda_1 = 0$, $\lambda_2 \neq 0$.
- "lasso+fused": lasso penalty and fused lasso penalty. $\alpha = 1$, $\lambda_1 \neq 0$, $\lambda_2 \neq 0$.
- "GL+fused": group lasso penalty and fused lasso penalty. $\alpha = 0$, $\lambda_1 \neq 0$, $\lambda_2 \neq 0$.
- "SGL+fused": sparse group lasso penalty and fused lasso penalty. $\alpha \neq 0$ or 1, $\lambda_1 \neq 0$, $\lambda_2 \neq 0$.

For `mpersonalized_cv`, `penalty` could take every option above except `"none"` since there is no need to do cross validation without regularization term.

As an example, we fit a SGL penalty with different rules by specifying `single_rule = FALSE` and `penalty = "SGL"`. `coef` will return a coefficients matrix with rows indicating studies or outcomes and columns indicating covariates.

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

multiple_mod = mpersonalized_cv(problem = "meta-analysis",
                                Xlist = sim_dat$Xlist,
                                Ylist = sim_dat$Ylist,
                                Trtlist = sim_dat$Trtlist,
                                penalty = "SGL",
                                single_rule = FALSE)
str(coef(multiple_mod))
```

```
##  num [1:6, 1:51] 0.425 0.459 0.446 0.414 0.492 ...
```

```
set.seed(NULL)
```

5.2 `single_rule = TRUE`

If a single rule $g(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$ is fitted across all studies or outcomes, the objective function of optimization is instead formulated as

$$\min_g \frac{1}{2} \sum_{k=1}^{K} \sum_{i=1}^{n_k} \frac{|\hat{C}_k(X_{ki})|}{\sum_{i=1}^{n_k} |\hat{C}_k(X_{ki})|} \left[1\{\hat{C}_k(X_{ki}) > 0\} - g(X_{ki})\right]^2 + h(g) \tag{5}$$

for meta-analysis and

$$\min_g \frac{1}{2} \sum_{k=1}^{K} \sum_{i=1}^{n} \frac{|\hat{C}_k(X_i)|}{\sum_{i=1}^{n} |\hat{C}_k(X_i)|} \left[1\{\hat{C}_k(X_i) > 0\} - g(X_i)\right]^2 + h(g) \tag{6}$$

for multiple outcomes. The regularization term h is simply a lasso penalty

$$h = \lambda_s \|\beta\|_1. \tag{7}$$

If `single_rule = TRUE` when using `mpersonalized` and `mpersonalized_cv`, users again specify the `penalty` argument and then either use the default sequence computed by the function or specify a sequence. For `mpersonalized`, the argument `penalty` could be chosen from:

- "none": no penalty. $\lambda_{single} = 0$.
- "lasso": lasso penalty. $\lambda_{single} \neq 0$.

For `mpersonalized_cv`, the only option is `"lasso"`.

As an example, we could fit a single rule with lasso penalty. `coef` will then return a vector of coefficients of the single rule.

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

single_mod = mpersonalized_cv(problem = "meta-analysis",
                              Xlist = sim_dat$Xlist,
                              Ylist = sim_dat$Ylist,
                              Trtlist = sim_dat$Trtlist,
                              penalty = "lasso",
                              single_rule = TRUE)

str(coef(single_mod))
```

```
##  Named num [1:51] 0.4406 0.1204 0.0702 0.1143 -0.117 ...
##  - attr(*, "names")= chr [1:51] "" "V1" "V2" "V3" ...
```

```
set.seed(NULL)
```

5.3 Multiple Rules vs. Single Rule

Using multiple rules will surely capture the individual features of studies or outcomes better so that we have a better understanding of individual study or outcome. However, given a new subejct, multiple rules are likely to give contrary recommendations, which could be difficult to implemented in practice. On the other hand, if a single rule is used across all the studies or outcomes, we obtain a more consistent and general rule with the price of certian flexibility. Which formulation is better really depends on the goal of the research and we leave the decision to users.

6 Penalty Parameter Sequence

In this section, we will cover more details of the penalty parameter sequence used in `mpersonalized` and `mpersonalized_cv`. Both functions offer a default penalty parameter sequence, regardless of `single_rule` and `penalty`. The default sequence for `lambda1` is computed through `SGL` package, for `lambda2` computed through `genlasso` and for `single_rule_lambda` computed through `glmnet`. When both `lambda1` and `lambda2` are involved in the penalty, then the penalty parameter sequence is the grid of default `lambda1` sequence and default `lambda2` sequence.

6.1 User-specified Sequence

Users can always supply a specified sequence of penalty parameter if the default one is not desired. The user-specified sequence is passed to the functions through arguments `lambda1`, `lambda2`, `alpha` and `single_rule_lambda`, which corresponds to $\lambda 1$, $\lambda 2$, α in 4, and λ_s in 7. Depending on the values of `single_rule` and `penalty`, these functions will detect whether users have sepcified any corresponding penalty parameter.

For example,

```
set.seed(123)
# Simulated dataset for meta-analysis
```

```r
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

multiple_mod = mpersonalized(problem = "meta-analysis",
                             Xlist = sim_dat$Xlist,
                             Ylist = sim_dat$Ylist,
                             Trtlist = sim_dat$Trtlist,
                             penalty = "lasso",
                             lambda1 = seq(0.01, 0.05, 0.01),
                             single_rule = FALSE)

multiple_mod$penalty_parameter_sequence
```

```
##      lambda1
## [1,]    0.01
## [2,]    0.02
## [3,]    0.03
## [4,]    0.04
## [5,]    0.05
```

```r
single_mod = mpersonalized(problem = "meta-analysis",
                           Xlist = sim_dat$Xlist,
                           Ylist = sim_dat$Ylist,
                           Trtlist = sim_dat$Trtlist,
                           penalty = "lasso",
                           single_rule_lambda = seq(0.01, 0.05, 0.01),
                           single_rule = TRUE)

single_mod$penalty_parameter_sequence
```

```
##      single_rule_lambda
## [1,]               0.01
## [2,]               0.02
## [3,]               0.03
## [4,]               0.04
## [5,]               0.05
```

```r
set.seed(NULL)
```

If, however, users supply the sequence to wrong arguments given `single_rule` and `penalty`, the functions will then use default computed one and corresponding warning message will be generated. For example, when `single_rule = TRUE` and `penalty = "lasso"`, users specify the `lambda1` argument rather than `single_rule_lambda`.

```r
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

single_mod = mpersonalized(problem = "meta-analysis",
                           Xlist = sim_dat$Xlist,
                           Ylist = sim_dat$Ylist,
                           Trtlist = sim_dat$Trtlist,
                           penalty = "lasso",
                           lambda1 = seq(0.01, 0.05, 0.01),
                           single_rule = TRUE)
```

```
## Warning in mpersonalized(problem = "meta-analysis", Xlist = sim_dat
```

```
## $Xlist, : When single rule = TRUE, the value for lambda1, lambda2, alpha
## are ignored!
```

```
# lambda1 is ignored and default single_rule_lambda is used
head(single_mod$penalty_parameter_sequence)
```

```
##      single_rule_lambda
## [1,]          1.2181456
## [2,]          1.0094076
## [3,]          0.8364383
## [4,]          0.6931085
## [5,]          0.5743394
## [6,]          0.4759222
```

```
set.seed(NULL)
```

Or when `single_rule = TRUE` and `penalty = "none"`, in which case users do not need to specify any penalty parameter,

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

single_mod = mpersonalized(problem = "meta-analysis",
                           Xlist = sim_dat$Xlist,
                           Ylist = sim_dat$Ylist,
                           Trtlist = sim_dat$Trtlist,
                           penalty = "none",
                           single_rule_lambda = seq(0.01, 0.05, 0.01),
                           single_rule = TRUE)
```

```
## Warning in mpersonalized(problem = "meta-analysis", Xlist = sim_dat
## $Xlist, : When single rule = TRUE and penalty = none, the value for
## single_rule_lambda are ignored!
```

```
# lambda1 is ignored
head(single_mod$penalty_parameter_sequence)
```

```
## NULL
```

```
set.seed(NULL)
```

Or when `single_rule = FALSE` and `penalty = "lasso"`, *alpha* will be foreced to be 1,

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

multiple_mod = mpersonalized(problem = "meta-analysis",
                           Xlist = sim_dat$Xlist,
                           Ylist = sim_dat$Ylist,
                           Trtlist = sim_dat$Trtlist,
                           penalty = "lasso",
                           alpha = 0.5,
                           single_rule = FALSE)
```

```
## Warning in mpersonalized(problem = "meta-analysis", Xlist = sim_dat
## $Xlist, : When penalty = lasso, alpha is automatically set to be 1!
```

```r
# alpha is forced to be 1
multiple_mod$alpha
```

```
## [1] 1
```

```r
set.seed(NULL)
```

These warning messages should help users to understand the framework better, and meanwhile the functions automatically correct the errors.

6.2 User-specified Length of Sequence

Sometimes users may not know what values should be supplied to the sequence, but they do want to control the length of the sequence. This could be achieved through arguments `num_lambda1`, `num_lambda2` and `num_single_rule_lambda`. However, if user do specify a sequence, the corresponding length will be ignored.

We could, for example, set the length of `lambda1` sequence to 20.

```r
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

multiple_mod = mpersonalized(problem = "meta-analysis",
                             Xlist = sim_dat$Xlist,
                             Ylist = sim_dat$Ylist,
                             Trtlist = sim_dat$Trtlist,
                             penalty = "lasso",
                             num_lambda1 = 20,
                             single_rule = FALSE)

length(multiple_mod$penalty_parameter_sequence)
```

```
## [1] 20
```

```r
set.seed(NULL)
```

Note when `single_rule = TRUE` and an integer is assinged `num_single_rule_lambda`, the actual length of penalty parameter sequence used might be actually shorter than `num_single_rule_lambda`. This is because `mpersonalized` uses `glmnet` package for numerical computation when `single_rule = TRUE`, and the path algorithm will stop early if the percent deviance explained does not change sufficently from one lambda to the next.

When `single_rule = FALSE`, numerical computation relies on `SGL` package and the alternating direction method of multipliers (ADMM) algorithm. Specifically, when `penalty` is `"lasso"`, `"SGL"` or `"GL"`, the optimization problem can be numerically solved by `SGL` package; when `penalty` is `"fused"`, `"lasso+fused"`, `"GL+fused"` or `"SGL+fused"`, we have to rely on the ADMM algorithm. The ADMM algorithm is written in `mpersonalized` package with original code and users can change parameters of the algorithm through `admm_control` argument in `mpersonalized` and `mpersonalized`. More details of `admm_control` will be explained in appendix. Since the ADMM algorithm is written in R code, the computation will be slow when `penalty` involves a fused lasso part. Therefore, we recommend using a short sequence if fused lasso term is involved. The computation should be able to speed up if the corresponding ADMM algorithm is implemented in a more low-level programming language, e.g. C++.

So far We have covered the arguments needed to fit a model via `mpersonalize` or `mpersonalized`. After the fitting process,`mpersonalized` will return an object of class `"mp"`. The return object contains all the information of the fitted models. Each value in the penalty parameter sequence has a corresponding fitted

model. `mpersonalized_cv` will return an obejct of class `"mp_cv"`, which only contains the fitted model corresponding to the optimal penalty parameter value.

Users can directly access the fitted model from class `"mp"` and `"mp_cv"` objects. For example, the coefficients of some `"mp_cv"` class object could be obtained through `mp_cv$intercept` and `mp_cv$beta`. Alternatively, users could also apply S3 methods to these objects. We will talk more about the available S3 methods for `"mp"` and `"mp_cv"` in next section.

7 S3 Methods for Class "mp" and "mp_cv"

S3 methods provide a convenient way to extract model information from S3 objects. In `mpersonalized`, a few methods are included for further analysis of the fitted model.

7.1 coef

Just like most `coef` methods, `coef` for `"mp"` and `"mp_cv"` classes will extract coefficients from the fitted models. For `"mp"` classes, `coef` will return a list object and each element in the list denotes the coefficients corresponding to one value in the penalty parameter sequence. For `"mp_cv"` classes, `coef` will return the coefficients corresponding to the optimal penalty parameter value. The return coefficients will be in matrix form if `single_rule = FALSE` is used in fitting and in vector if `single_rule = TRUE`.

To obtain the coefficients corresponding to some specific penalty parameter value, users can use its index in penalty parameter sequence.

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

mod = mpersonalized(problem = "meta-analysis",
                    Xlist = sim_dat$Xlist,
                    Ylist = sim_dat$Ylist,
                    Trtlist = sim_dat$Trtlist,
                    penalty = "SGL",
                    single_rule = FALSE)

mod$penalty_parameter_sequence
```

```
##          lambda1
##  [1,] 0.28788909
##  [2,] 0.22290207
##  [3,] 0.17258498
##  [4,] 0.13362628
##  [5,] 0.10346197
##  [6,] 0.08010685
##  [7,] 0.06202382
##  [8,] 0.04802279
##  [9,] 0.03718231
## [10,] 0.02878891
```

```
# lambda1 = 0.173 is the 3rd value in the sequence, to obtain corresponding coefficients
coef(mod)[[3]][,1:5]
```

```
##      intercept
## [1,] 0.4213146 0.05858765 0 0.000000000  0.00000000
```

```
## [2,] 0.4468346 0.05300223 0 0.071623569  0.00000000
## [3,] 0.4158377 0.05253169 0 0.015880690 -0.03091217
## [4,] 0.3771854 0.00000000 0 0.027045536 -0.06469979
## [5,] 0.4772659 0.00000000 0 0.007493161 -0.02526791
## [6,] 0.3594218 0.00000000 0 0.000000000 -0.07901545
```

```
cv_mod = mpersonalized_cv(problem = "meta-analysis",
                          Xlist = sim_dat$Xlist,
                          Ylist = sim_dat$Ylist,
                          Trtlist = sim_dat$Trtlist,
                          penalty = "SGL",
                          single_rule = FALSE)

# with cross validation, coef will directly return the coefficients with the optimal penalty parameter
coef(cv_mod)[1:5]
```

```
## [1] 0.4231731 0.4551443 0.4417547 0.4069381 0.4902162
```

```
set.seed(NULL)
```

7.2 predict

`predict` function predicts optimal treatments for individual subjects. If `newx` is additionally supplied to `predict` besides the `"mp"` or `"mp_cv"` object, predicition is made for new subejcts with covariates matrix represented by `newx`; otherwise, prediction is made for the original subejcts used in fitting procedure.

The difference between `"mp"` and `"mp_cv"` in `predict` method is still that for `"mp"`, `predict` returns predictions for every penalty parameter value in the sequence while for `"mp_cv"`, `predict` only returns the prediction using the optimal penalty paratmer value.

For every specific penalty parameter value, `predict` returns a matrix of optimal treatments and also a matrix of benefit score, that is, $g_1(X), \ldots, g_K(X)$, with rows indicating subjects and columns indicating stuides or outcomes if `newx` is supplied. In addition, `predict` provides an extra argument `overall_rec` indicating whether a wegihted voting of the individual recommendations from all studies or outcomes should also be computed. If `overall_rec = TRUE`, weights for each voter can be further specified by users and the overall recommendation will be added to the optimal treatments matrix.

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

cv_mod = mpersonalized_cv(problem = "meta-analysis",
                          Xlist = sim_dat$Xlist,
                          Ylist = sim_dat$Ylist,
                          Trtlist = sim_dat$Trtlist,
                          penalty = "SGL",
                          single_rule = FALSE)

# simulate the covariates matrix for 10 new subjects
# overall_rec = TRUE by default
newx = matrix(rnorm(10 * 50), nrow = 10)
predict(cv_mod, newx = newx)$opt_treatment
```

```
##       Study 1 Study 2 Study 3 Study 4 Study 5 Study 6 Overall Rec
##  [1,]       0       0       0       0       0       0           0
##  [2,]       0       0       0       0       0       0           0
```

```
##  [3,]       0       0       0       0       0       0           0
##  [4,]       0       0       0       0       0       0           0
##  [5,]       1       1       0       0       0       0           0
##  [6,]       0       1       0       0       0       0           0
##  [7,]       0       0       0       0       0       0           0
##  [8,]       0       0       0       0       0       0           0
##  [9,]       0       0       1       1       1       1           1
## [10,]       1       1       1       1       1       0           1
```

```
round(predict(cv_mod, newx = newx)$benefit_score,3)
```

```
##       Study 1 Study 2 Study 3 Study 4 Study 5 Study 6
##  [1,]  -0.256   0.257  -0.034   0.225   0.072   0.030
##  [2,]   0.279   0.035  -0.007  -0.110   0.045  -0.088
##  [3,]   0.080   0.327   0.228   0.272   0.049   0.086
##  [4,]   0.001   0.370   0.182   0.342   0.189   0.164
##  [5,]   0.768   0.572   0.483   0.147   0.494   0.193
##  [6,]   0.066   0.686   0.320   0.444   0.330   0.212
##  [7,]   0.334   0.465   0.445   0.358   0.369   0.261
##  [8,]   0.178   0.241   0.198   0.162   0.081   0.069
##  [9,]   0.470   0.474   0.691   0.683   0.658   0.598
## [10,]   0.519   0.672   0.604   0.588   0.675   0.487
```

```
set.seed(NULL)
```

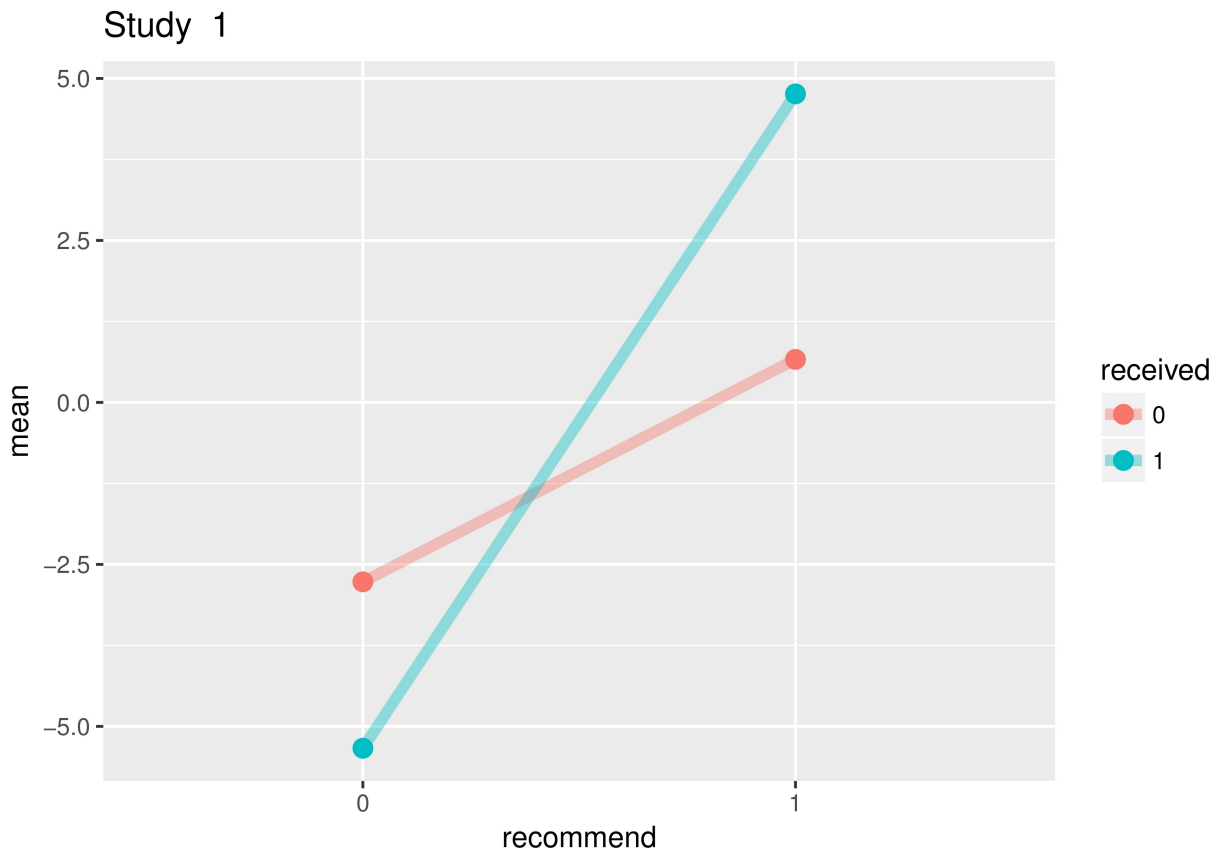
7.3 plot

`plot` method provides interaction plots for `"mp"` and `"mp_cv"` objects. More specifically, it plots the interaction between recommended treatment and received treatment for the fitted data.

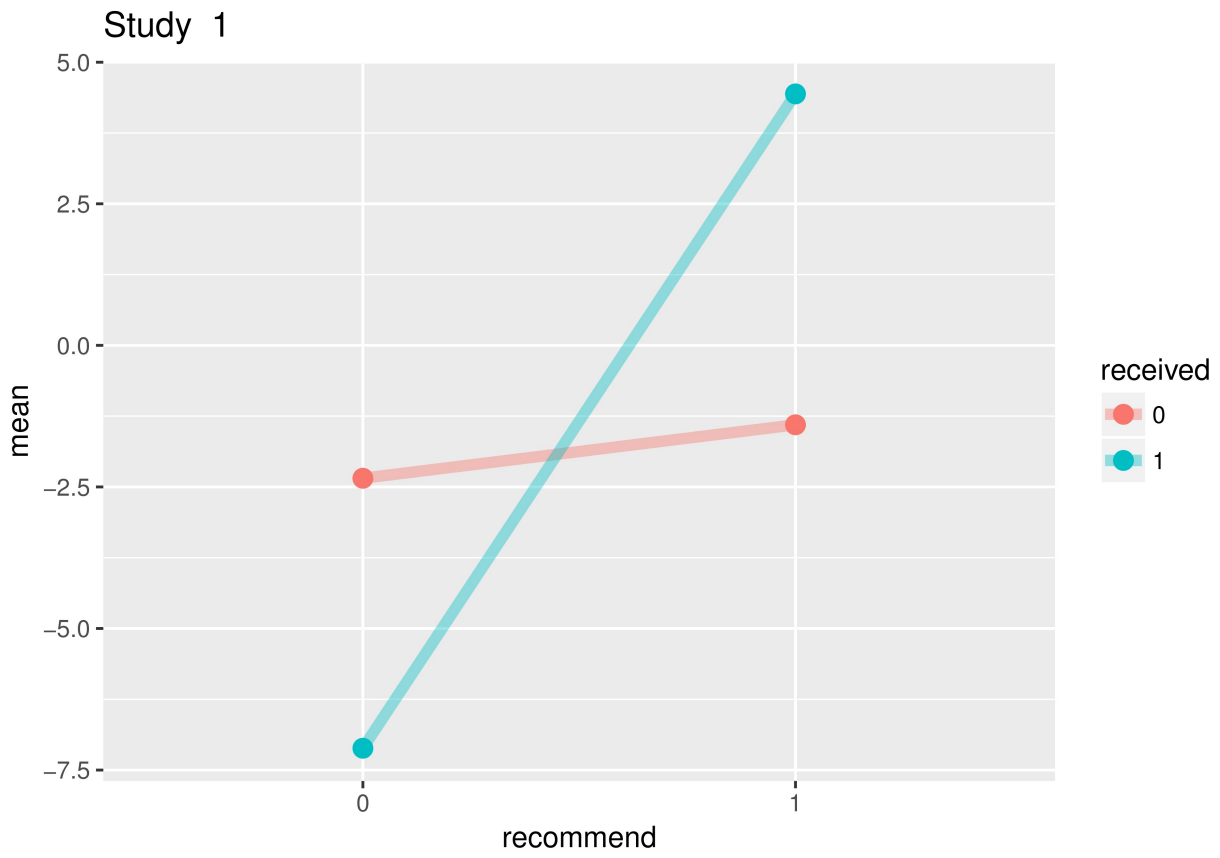
The interaction plots can be used to check if the estimated treatment rule is working as intended, at least in training set. Based on interactions of recommended treatment ($\hat{A}$) and received treatment (A), we divide subjects into four groups. Assuming large outcome is more favorable, then ideally the group with $\hat{A} = 1$ and $A = 1$ should have average higher outcome compared to the group with $\hat{A} = 1$ and $A = 0$; similarly, the group with $\hat{A} = 0$ and $A = 0$ should also have average higher outcome compared to the group with $\hat{A} = 0$ and $A = 1$.

For `"mp"` class, `plot` requires an extra argument `penalty_index`. Not returning the plots throughout the whole penalty parameter sequence is mainly due to computation memory consideration. For `"mp_cv"` class, `plot` again returns the interaction plots corresponding to the optimal penalty parameter value.

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

mod = mpersonalized(problem = "meta-analysis",
                    Xlist = sim_dat$Xlist,
                    Ylist = sim_dat$Ylist,
                    Trtlist = sim_dat$Trtlist,
                    penalty = "SGL",
                    single_rule = FALSE)

# interaction plot of study 1 corresponding to the model fitted with the 3rd penalty parameter value
plot(mod, penalty_index = 3)[[1]]
```




```
cv_mod = mpersonalized_cv(problem = "meta-analysis",
                          Xlist = sim_dat$Xlist,
                          Ylist = sim_dat$Ylist,
                          Trtlist = sim_dat$Trtlist,
                          penalty = "SGL",
                          single_rule = FALSE)

# interaction plot of study 1
plot(cv_mod)[[1]]
```




```
set.seed(NULL)
```

8 Appendix

Other than the options mentioned above, some internal arguments are also provided to control more subtle aspects in the whole modeling procedure. Typically, users do not need to specify them but we will cover it here in cases needed.

8.1 Internal Arguments for Contrast Estimation

In `mpersonalized` and `mpersonalized_cv`, contrast function for each study or outcome need to be estimated ahead of the optimization problem being formed. One of the simplest estimators for contrast function is the inverse probability weighted estimator,

$$\hat{C}(X) = \frac{YA}{P(X)} - \frac{Y(1-A)}{1-P(X)}, \tag{8}$$

where $P(X)$ is the propensity score.

Although it is unbiased, the inverse probability weighted estimator is usually not efficient. The efficiency could be improved by subtracting the outcome with an efficiency augmentation term $a(X)$,

$$a(X) = \{1 - P(X)\}E(Y|A = 1, X) + P(X)E(Y|A = 0, X). \tag{9}$$

The new estimator is refered to as augmented inverse probability weighted estimaator (AIPWE) (Robins, Rotnitzky, and Zhao 1994). This efficiency augmentation term requires an estimate of the outcome model $E(Y|A=1,X)$ and $E(Y|A=0,X)$. Note here we merely use the outcome model to increase efficiency rather than directly predict the optimal treatment. Hence, it does not necessarily have to be very accurate.

To control arguments related to contrast estimation, users could supply `contrast_builder_control` in `mpersonalized` and `mpersonalized_cv`. `contrast_builder_control` should be a list object with the following optional elements:

- eff_aug: a logical value denoting whether efficiency augmentation is carried out. Default value is TRUE.
- response_model: a character string specify what model we should use to estimate $E(Y|A=1,X)$ and $E(Y|A=0,X)$. Options are "lasso" and "linear". Default is "lasso".
- contrast_builder_folds: a integer specifying the cross validation folds to use when `resposne_model = "lasso"`.

Although `response_model` only offers "lasso" and "linear" options, users could still implement a more complicated model by manually subtracting the augmentation term from outcome before suppply it to `mpersonalized` and `mpersonalized_cv`, and then set `eff_aug = FALSE`.

An example of efficiency agumentation with lasso for outcome modeling:

```
set.seed(123)
# Simulated dataset for meta-analysis
sim_dat = simulated_dataset(n = 200, problem = "meta-analysis")

single_mod = mpersonalized_cv(problem = "meta-analysis",
                              Xlist = sim_dat$Xlist,
                              Ylist = sim_dat$Ylist,
                              Trtlist = sim_dat$Trtlist,
                              penalty = "lasso",
                              single_rule = TRUE,
                              contrast_builder_control = list(eff_aug = TRUE,
                                                              response_model = "lasso"))

set.seed(NULL)
```

8.2 Internal Argument for ADMM

When fused lasso penalty is involved, our package relis on ADMM algorithm (Boyd et al. 2011) for numerical optimization. For a general optimization problem:

$$\begin{aligned} &\text{minimize} && f(x)+g(z) && (10)\\ &\text{subject to} && Ax+Bz=c && (11)\end{aligned}$$

The corresponding augmented Langrangian problem is constructed as

$$L_\rho(x,z,y)=f(x)+g(z)+y^T(Ax+Bz-c)+(\rho/2)\|Ax+Bz-c\|_2^2 \quad (12)$$

and ADMM iterates through the following steps

$$x^{k+1} := \arg\min_x L_\rho(x, z^k, y^k) \tag{13}$$
$$z^{k+1} := \arg\min_z L_\rho(x^{k+1}, z, y^k) \tag{14}$$
$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c), \tag{15}$$

where $\rho > 0$ and called augmented Lagrangian parameter.

The stopping criteria of ADMM algorithm requires both the dual residual

$$s^{k+1} = \rho A^T B(z^{k+1} - z^k) \tag{16}$$

and the primal residual

$$r^{k+1} = Ax^{k+1} + Bz^{k+1} - c \tag{17}$$

to be small.

Given absolute tolerance $\epsilon^{\text{abs}} > 0$ and relative tolerance $\epsilon^{\text{rel}} > 0$, ADMM stops when $\|r^k\|_2 \leq \epsilon^{\text{pri}}$ and $\|s^k\|_2 \leq \epsilon^{\text{dual}}$ are both satisfied, where

$$\epsilon^{\text{pri}} = \sqrt{p}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\|AX^k\|_2, \|Bz^k\|_2, \|c\|_2\}, \tag{18}$$
$$\epsilon^{\text{dual}} = \sqrt{n}\epsilon^{abs} + \epsilon^{rel}\|A^T y^k\|_2. \tag{19}$$

`mpersonalized` and `mpersonalized` provides the argument `admm_control` for users to control the internal parameters of ADMM algorithm. Similar to `contrast_builder_control`, `admm_control` should be a list which could takes the following options:

- abs.tol: absolute tolerance ϵ^{abs}. Default value is `1e-5`.
- rel.tol: relative tolerance ϵ^{rel}. Default value is `1e-5`.
- maxit: maximum number of iterations in ADMM algorithm. Default value is 500.
- rho: the Lagrangian parameter. Default value depends on the data input.

Note that large values of ρ tend to produce small primal residuals. Conversely, small values of ρ tend to reduce the dual residual, but at the expense of resulting in a larger primal residual. It is hard to select a proper ρ manually and we recommend users to leave it to the program to estimate the optimal ρ by default.

References

Boyd, Stephen, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers." *Foundations and Trends® in Machine Learning* 3 (1): 1–122. doi:10.1561/2200000016.

Robins, James M., Andrea Rotnitzky, and Lue Ping Zhao. 1994. "Estimation of Regression Coefficients When Some Regressors Are Not Always Observed." *Journal of the American Statistical Association* 89 (427): 846–66.

Zhang, B., A. A. Tsiatis, M. Davidian, M. Zhang, and E. Laber. 2012. "Estimating Optimal Treatment Regimes from a Classification Perspective." *Stat* 1 (1). Wiley Online Library: 103–14.