OtherDetails

```
#> Loading required package: ptLasso
#> Loading required package: ggplot2
#> Loading required package: glmnet
#> Loading required package: Matrix
#> Loaded glmnet 4.1-8
#> Loading required package: gridExtra
```

0.1 Encoding the `groups` parameter

The `groups` parameter is a vector with one entry for each observation, and for data with k groups, the groups must be encoded as integers from 1 to k. So, if we have 3 groups that are originally encoded as strings:

```r
groups.strings = sample(c("group1", "group2", "group3"), 10, replace = TRUE)
```

We could convert them to integers (1 through 3) as follows:

```r
groups.ints = rep(1, 10)
groups.ints[groups.strings == "group2"] = 2
groups.ints[groups.strings == "group3"] = 3
```

Now, they are in the right form for calling `ptLasso` (or `cv.ptLasso`).

0.2 Choosing α, the pretraining hyperparameter

Selecting the hyperparameter α is an important part of pretraining. The simplest way to do this is to use `cv.ptLasso` – this will automatically perform pretraining using $\alpha = 0, 0.1, 0.2, \ldots, 1$. It additionally returns the CV performance estimates for each value of α:

```r
cvfit <- cv.ptLasso(x, y, groups)
cvfit
#> 
#> Call:
#> cv.ptLasso(x = x, y = y, groups = groups, family = "gaussian",
#>     type.measure = "mse", use.case = "inputGroups")
#> 
#> 
#> 
#> type.measure:  mse
#> 
#> 
#>            alpha overall  mean wtdMean group1 group2 group3 group4 group5
#> Overall              691.0 691.0   691.0  730.1  497.9  554.0  664.8 1008.0
#> Pretrain     0.0   544.0 544.0   544.0  486.1  505.3  558.4  537.1  633.0
#> Pretrain     0.1   523.8 523.8   523.8  484.0  492.5  565.5  498.4  578.7
#> Pretrain     0.2   519.9 519.9   519.9  448.6  467.0  604.6  501.2  578.2
#> Pretrain     0.3   501.8 501.8   501.8  431.8  466.9  587.8  478.4  544.3
#> Pretrain     0.4   504.7 504.7   504.7  433.0  477.4  567.6  483.9  561.9
#> Pretrain     0.5   498.5 498.5   498.5  384.4  464.0  582.1  488.4  573.4
#> Pretrain     0.6   506.3 506.3   506.3  379.3  478.5  595.1  496.4  582.2
```

```
#> Pretrain     0.7   509.8 509.8   509.8  393.0  469.7  620.1  494.3  571.7
#> Pretrain     0.8   515.2 515.2   515.2  414.4  479.4  604.1  470.9  607.4
#> Pretrain     0.9   511.0 511.0   511.0  384.7  485.9  600.7  505.3  578.4
#> Pretrain     1.0   527.7 527.7   527.7  377.8  495.5  639.2  510.2  615.7
#> Individual         527.7 527.7   527.7  377.8  495.5  639.2  510.2  615.7
#>
#> alphahat (fixed) = 0.5
#> alphahat (varying):
#> group1 group2 group3 group4 group5
#>    1.0    0.5    0.0    0.8    0.3
```

Of course, you can specify the values of α to consider:

```
cvfit <- cv.ptLasso(x, y, groups, alphalist = c(0, 0.5, 1))
cvfit
#>
#> Call:
#> cv.ptLasso(x = x, y = y, groups = groups, alphalist = c(0, 0.5,
#>     1), family = "gaussian", type.measure = "mse", use.case = "inputGroups")
#>
#>
#>
#> type.measure:  mse
#>
#>
#>             alpha overall  mean wtdMean group1 group2 group3 group4 group5
#> Overall             704.1 704.1   704.1  742.2  502.8  571.4  670.5 1033.7
#> Pretrain     0.0   555.7 555.7   555.7  463.8  519.8  608.3  566.6  620.1
#> Pretrain     0.5   503.9 503.9   503.9  387.5  492.1  594.2  485.2  560.2
#> Pretrain     1.0   531.5 531.5   531.5  397.7  516.3  631.4  530.0  582.0
#> Individual         531.5 531.5   531.5  397.7  516.3  631.4  530.0  582.0
#>
#> alphahat (fixed) = 0.5
#> alphahat (varying):
#> group1 group2 group3 group4 group5
#>    0.5    0.5    0.5    0.5    0.5
```

At prediction time, `cv.ptLasso` will automatically use the α that had the best CV performance on average across all groups:

```
predict(cvfit, xtest, groupstest, ytest=ytest)
#>
#> Call:
#> predict.cv.ptLasso(cvfit = cvfit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.5
#>
#> Performance (Mean squared error):
#>
#>             allGroups  mean group1 group2 group3 group4 group5    r^2
#> Overall         757.1 757.1  815.7  542.6  567.1  792.7 1067.5 0.5362
#> Pretrain        498.1 498.1  549.4  431.0  532.5  501.5  475.8 0.6949
#> Individual      528.8 528.8  584.1  441.7  567.2  548.0  503.3 0.6760
```

```
#>
#> Support size:
#>
#> Overall    50
#> Pretrain   98 (20 common + 78 individual)
#> Individual 108
```

But we could instead choose to use a different α for each group – `cv.ptLasso` already figured out which α has the best CV performance for each group. To use group-specific values of α, specify `alphatype = "varying"` at prediction time:

```
predict(cvfit, xtest, groupstest, ytest=ytest, alphatype = "varying")
#>
#> Call:
#> predict.cv.ptLasso(cvfit = cvfit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest, alphatype = "varying")
#>
#>
#> alpha:
#> group1 group2 group3 group4 group5
#>    0.5    0.5    0.5    0.5    0.5
#>
#>
#> Performance (Mean squared error):
#>             overall  mean wtdMean group1 group2 group3 group4 group5
#> Overall       757.1 757.1   757.1  815.7  542.6  567.1  792.7 1067.5
#> Pretrain      498.1 498.1   498.1  549.4  431.0  532.5  501.5  475.8
#> Individual    528.8 528.8   528.8  584.1  441.7  567.2  548.0  503.3
#>
#>
#> Support size:
#>
#> Overall    50
#> Pretrain   98 (20 common + 78 individual)
#> Individual 108
```

0.3 Choosing λ, the lasso hyperparameter, for the first stage of pretraining

The first step of pretraining fits the overall model with `cv.glmnet` and selects a model along the λ path. The second stage uses this model's support and predictions to train the group-specific models.

So, at train time, we need to know which value of λ to use for the first stage. This can be specified in `ptLasso` with the argument `overall.lambda`:

```
fit <- ptLasso(x, y, groups, alpha = 0.5, overall.lambda = "lambda.1se")
```

The default value of `overall.lambda` is "lambda.1se", as we found this to offer slightly better performance. However, other good choices include "lambda.min" or a numeric value (as in `predict.cv.glmnet`). For example:

```
fit <- ptLasso(x, y, groups, alpha = 0.5, overall.lambda = "lambda.min")
```

Whatever choice is made at train time will be automatically used at test time, and this cannot be changed. (The fitted model from the second stage of pretraining expects the offset to have been computed using a particular model – it does not make sense to compute the offset using a different model with a different λ!)

0.4 Printing progress during model training

When models take a long time to train, it can be useful to print out progress during training. `ptLasso` has two ways to do this (and they can be combined). First, we can simply print out which model is being fitted using `verbose = TRUE`:

```
fit <- ptLasso(x, y, groups, alpha = 0.5, verbose = TRUE)
#> Fitting overall model
#> Fitting individual models
#>  Fitting individual model 1 / 5
#>  Fitting individual model 2 / 5
#>  Fitting individual model 3 / 5
#>  Fitting individual model 4 / 5
#>  Fitting individual model 5 / 5
#> Fitting pretrained lasso models
#>  Fitting pretrained model 1 / 5
#>  Fitting pretrained model 2 / 5
#>  Fitting pretrained model 3 / 5
#>  Fitting pretrained model 4 / 5
#>  Fitting pretrained model 5 / 5
```

We can also print out a progress bar for *each model* that is being fit – this functionality comes directly from `cv.glmnet`, and follows its notation. (To avoid cluttering this document, we do not run the following example.)

```
fit <- ptLasso(x, y, groups, alpha = 0.5, trace.it = TRUE)
```

And of course, we can combine these to print out (1) which model is being trained and (2) the corresponding progress bar.

```
fit <- ptLasso(x, y, groups, alpha = 0.5, verbose = TRUE, trace.it = TRUE)
```

0.5 Using individual and overall models that have already been trained

`ptLasso` will fit the overall and individual models. However, if you have already trained the overall or individual models, you can pass these directly to `ptLasso` and avoid refitting them.

Here is an example. We will fit an overall model and individual models, and then we will show how to pass them to `ptLasso`. Importantly, we specify `keep = TRUE` when fitting these models for two reasons: (1) `ptLasso` uses prevalidated predictions from the overall model for the second stage of pretraining, and (2) we compute CV performance using the prevalidated predictions.

```
overall.model = cv.glmnet(x, y, keep = TRUE)
individual.models = lapply(1:5,
                           function(kk) cv.glmnet(x[groups == kk, ],
                                                  y[groups == kk],
                                                  keep = TRUE))
```

Here is how we would pass these trained models through to `ptLasso`. Using `verbose = TRUE` shows us what models are being trained (and confirms that we are not refitting the overall and individual models).

```
fit <- ptLasso(x, y, groups,
               fitoverall = overall.model,
               fitind = individual.models,
               verbose = TRUE)
```

Of course we could pass just the overall or individual models to ‘ptLasso:

```r
fit <- ptLasso(x, y, groups, fitoverall = overall.model, verbose = TRUE)
#> Fitting individual models
#>  Fitting individual model 1 / 5
#>  Fitting individual model 2 / 5
#>  Fitting individual model 3 / 5
#>  Fitting individual model 4 / 5
#>  Fitting individual model 5 / 5
#> Fitting pretrained lasso models
#>  Fitting pretrained model 1 / 5
#>  Fitting pretrained model 2 / 5
#>  Fitting pretrained model 3 / 5
#>  Fitting pretrained model 4 / 5
#>  Fitting pretrained model 5 / 5
```

```r
fit <- ptLasso(x, y, groups, fitind = individual.models, verbose = TRUE)
#> Fitting overall model
#> Fitting pretrained lasso models
#>  Fitting pretrained model 1 / 5
#>  Fitting pretrained model 2 / 5
#>  Fitting pretrained model 3 / 5
#>  Fitting pretrained model 4 / 5
#>  Fitting pretrained model 5 / 5
```

0.6 Fitting the overall model without group-specific intercepts

When we fit the overall model with input grouped data, we solve the following:

$$\hat{\mu_0}, \hat{\theta}_2, \dots, \hat{\theta}_k, \hat{\beta}_0 = \arg \min_{\mu, \theta_2, \dots, \theta_k, \beta} \frac{1}{2} \sum_{k=1}^{K} \| y_k - (\mu \mathbf{1} + \theta_k \mathbf{1} + X_k \beta) \|_2^2 + \lambda ||\beta||_1, \tag{1}$$

where $\hat{\theta_1}$ is defined to be 0. If this is not desired, we can instead fit the following:

$$\hat{\mu_0}, \hat{\beta}_0 = \arg \min_{\mu, \beta} \frac{1}{2} \sum_{k=1}^{K} \| y_k - (\mu \mathbf{1} + X_k \beta) \|_2^2 + \lambda ||\beta||_1. \tag{2}$$

This may be useful in settings where the groups are different between train and test sets and we show an example in the section "Different groups in train and test data". To do this, use the argument `group.intercepts = FALSE`. In our toy example, omitting the group-specific intercepts results in slightly worse CV performance; we expect this to be the case more generally.

```r
cvfit <- cv.ptLasso(x, y, groups, group.intercepts = FALSE)
cvfit
#>
#> Call:
#> cv.ptLasso(x = x, y = y, groups = groups, group.intercepts = FALSE,
#>     family = "gaussian", type.measure = "mse", use.case = "inputGroups")
#>
#>
#>
#> type.measure:  mse
#>
#>
#>             alpha overall  mean wtdMean group1 group2 group3 group4 group5
#> Overall              694.0 694.0   694.0  717.8  505.1  571.8  659.7 1015.4
```

```
#> Pretrain     0.0   593.4 593.4   593.4  519.6  499.2  583.4  617.4  747.5
#> Pretrain     0.1   511.5 511.5   511.5  429.9  442.8  604.6  500.3  580.1
#> Pretrain     0.2   490.3 490.3   490.3  388.1  452.3  592.0  457.1  561.8
#> Pretrain     0.3   499.2 499.2   499.2  392.5  469.0  580.6  473.6  580.5
#> Pretrain     0.4   507.4 507.4   507.4  401.7  474.3  599.0  488.4  573.5
#> Pretrain     0.5   517.3 517.3   517.3  388.6  479.9  600.1  528.0  589.9
#> Pretrain     0.6   513.8 513.8   513.8  383.0  493.8  574.7  497.0  620.3
#> Pretrain     0.7   518.0 518.0   518.0  384.8  490.4  597.2  533.3  584.4
#> Pretrain     0.8   516.9 516.9   516.9  416.9  472.2  601.2  485.2  609.0
#> Pretrain     0.9   517.9 517.9   517.9  423.7  478.9  599.8  498.9  588.0
#> Pretrain     1.0   515.9 515.9   515.9  402.3  498.3  606.3  467.9  604.8
#> Individual         515.9 515.9   515.9  402.3  498.3  606.3  467.9  604.8
#>
#> alphahat (fixed) = 0.2
#> alphahat (varying):
#> group1 group2 group3 group4 group5
#>    0.6    0.1    0.6    0.2    0.2
```

0.7 Arguments for use in `cv.glmnet`

Because model fitting is done with `cv.glmnet`, `ptLasso` can take and pass arguments to `glmnet`. Notable choices include `penalty.factor`, `weights`, `upper.limits`, `lower.limits` and `en.alpha` (known as `alpha` in `glmnet`). Importantly, `ptLasso` does not support the arguments `intercept`, `offset`, `fit` and `check.args`.

0.8 Parallelizing model fitting

For large datasets, we can parallelize model fitting within the calls to `cv.glmnet`. As in `cv.glmnet`, pass the argument `parallel = TRUE`, and register parallel beforehand:

```
require(doMC)
registerDoMC(cores = 4)
fit = ptLasso(x, y, groups = groups, family = "gaussian", type.measure = "mse", parallel=TRUE)
```