Input grouped data

```
require(ptLasso)
#> 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
```

Base case: input grouped data with a binomial outcome

In the Quick Start, we applied `ptLasso` to data with a continuous response. Here, we'll use data with a binary outcome. This creates a dataset with $k = 3$ groups (each with 100 observations), 5 shared coefficients, and 5 coefficients specific to each group.

```
set.seed(1234)

out = binomial.example.data()
x = out$x; y = out$y; groups = out$groups

outtest = binomial.example.data()
xtest = outtest$x; ytest = outtest$y; groupstest = outtest$groups
```

We can fit and predict as before. By default, `predict.ptLasso` will compute and return the *deviance* on the test set.

```
fit = ptLasso(x, y, groups, alpha = 0.5, family = "binomial")

predict(fit, xtest, groupstest, ytest = ytest)
#>
#> Call:
#> predict.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.5
#>
#> Performance (Deviance):
#>
#>            allGroups  mean wtdMean group_1 group_2 group_3
#> Overall        1.359 1.359   1.359   1.334   1.321   1.421
#> Pretrain       1.279 1.279   1.279   1.272   1.169   1.397
#> Individual     1.283 1.283   1.283   1.265   1.186   1.399
#>
#> Support size:
#>
#> Overall    7
#> Pretrain   12 (3 common + 9 individual)
```

```
#> Individual 20
```

We could instead compute the AUC by specifying the `type.measure` in the call to `ptLasso`. Note: `type.measure` is specified during model fitting and not prediction because it is used in each call to `cv.glmnet`.

```
fit = ptLasso(x, y, groups, alpha = 0.5, family = "binomial",
              type.measure = "auc")

predict(fit, xtest, groupstest, ytest = ytest)
#>
#> Call:
#> predict.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.5
#>
#> Performance (AUC):
#>
#>            allGroups   mean wtdMean group_1 group_2 group_3
#> Overall       0.6026 0.6039  0.6039  0.6161  0.6877  0.5080
#> Pretrain      0.6407 0.6524  0.6524  0.6936  0.7447  0.5190
#> Individual    0.6442 0.6618  0.6618  0.6936  0.7732  0.5186
#>
#> Support size:
#>
#> Overall    15
#> Pretrain   39 (3 common + 36 individual)
#> Individual 40
```

To fit the overall and individual models, we can use elasticnet instead of lasso by defining the parameter `en.alpha` (as in `glmnet` and described in the section "Fitting elasticnet or ridge models").

```
fit = ptLasso(x, y, groups, alpha = 0.5, family = "binomial",
              type.measure = "auc",
              en.alpha = .5)
predict(fit, xtest, groupstest, ytest = ytest)
#>
#> Call:
#> predict.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.5
#>
#> Performance (AUC):
#>
#>            allGroups   mean wtdMean group_1 group_2 group_3
#> Overall       0.6041 0.6018  0.6018  0.5928  0.6704  0.5422
#> Pretrain      0.6270 0.6547  0.6547  0.6781  0.7720  0.5141
#> Individual    0.6387 0.6598  0.6598  0.6756  0.7820  0.5218
#>
#> Support size:
#>
#> Overall    3
```

```
#> Pretrain   39 (3 common + 36 individual)
#> Individual 36
```

Using cross validation is the same as in the Gaussian case:

```
###################################################
# Fit:
###################################################
fit = cv.ptLasso(x, y, groups, family = "binomial", type.measure = "auc")
#> Warning: from glmnet C++ code (error code -100); Convergence for 100th lambda
#> value not reached after maxit=100000 iterations; solutions for larger lambdas
#> returned

#> Warning: from glmnet C++ code (error code -100); Convergence for 100th lambda
#> value not reached after maxit=100000 iterations; solutions for larger lambdas
#> returned
#> Warning: from glmnet C++ code (error code -92); Convergence for 92th lambda
#> value not reached after maxit=100000 iterations; solutions for larger lambdas
#> returned
#> Warning: from glmnet C++ code (error code -90); Convergence for 90th lambda
#> value not reached after maxit=100000 iterations; solutions for larger lambdas
#> returned

###################################################
# Predict with a common alpha for all groups:
###################################################
predict(fit, xtest, groupstest, ytest = ytest)
#>
#> Call:
#> predict.cv.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.7
#>
#> Performance (AUC):
#>
#>             allGroups   mean wtdMean group_1 group_2 group_3
#> Overall        0.5990 0.5960  0.5960  0.6030  0.6644  0.5206
#> Pretrain       0.6401 0.6640  0.6640  0.6965  0.7732  0.5222
#> Individual     0.6559 0.6707  0.6707  0.6936  0.7808  0.5377
#>
#> Support size:
#>
#> Overall    7
#> Pretrain   40 (3 common + 37 individual)
#> Individual 37

###################################################
# Predict with a different alpha for each group:
###################################################
predict(fit, xtest, groupstest, ytest = ytest, alphatype = "varying")
#>
#> Call:
```

```
#> predict.cv.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest, alphatype = "varying")
#>
#>
#> alpha:
#> group_1 group_2 group_3
#>     0.2     0.5     0.2
#>
#>
#> Performance (AUC):
#>            overall   mean wtdMean group_1 group_2 group_3
#> Overall     0.5990 0.5960  0.5960  0.6030  0.6644  0.5206
#> Pretrain    0.6359 0.6573  0.6573  0.6838  0.7736  0.5145
#> Individual  0.6559 0.6707  0.6707  0.6936  0.7808  0.5377
#>
#>
#> Support size:
#>
#> Overall    7
#> Pretrain   40 (3 common + 37 individual)
#> Individual 37
```

Base case: input grouped survival data

```
require(survival)
#> Loading required package: survival
```

Now, we will simulate survival times with 3 groups; the three groups have overlapping support, with 5 shared features and each has 5 individual features. To compute survival time, we start by computing survival $= X\beta + \epsilon$, where β is specific to each group and ϵ is noise. Because survival times must be positive, we modify this to be survival $=$ survival $+ 1.1 * \text{abs}(\text{min}(\text{survival}))$.

```
set.seed(1234)

n = 600; ntrain = 300
p = 50

x = matrix(rnorm(n*p), n, p)
beta1 = c(rnorm(5), rep(0, p-5))

beta2 = runif(p) * beta1 # Shared support
beta2 = beta2 + c(rep(0, 5), rnorm(5), rep(0, p-10)) # Individual features

beta3 = runif(p) * beta1 # Shared support
beta3 = beta3 + c(rep(0, 10), rnorm(5), rep(0, p-15)) # Individual features

# Randomly split into groups
groups = sample(1:3, n, replace = TRUE)

# Compute survival times:
survival = x %*% beta1
survival[groups == 2] = x[groups == 2, ] %*% beta2
survival[groups == 3] = x[groups == 3, ] %*% beta3
survival = survival + rnorm(n)
```

```r
survival = survival + 1.1 * abs(min(survival))

# Censoring times from a random uniform distribution:
censoring = runif(n, min = 1, max = 10)

# Did we observe surivival or censoring?
y = Surv(pmin(survival, censoring), survival <= censoring)

# Split into train and test:
xtest = x[-(1:300), ]
ytest = y[-(1:300), ]
groupstest = groups[-(1:300)]

x = x[1:300, ]
y = y[1:300, ]
groups = groups[1:300]
```

Training with `ptLasso` is much the same as it was for the continuous and binomial cases; the only difference is that we specify `family = "cox"`. By default, `ptLasso` uses the partial likelihood for model selection. We could instead use the C index.

```r
############################################################
# Default -- use partial likelihood as the type.measure:
############################################################
fit = ptLasso(x, y, groups, alpha = 0.5, family = "cox")
predict(fit, xtest, groupstest, ytest = ytest)
#>
#> Call:
#> predict.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.5
#>
#> Performance (Deviance):
#>
#>            allGroups  mean wtdMean group_1 group_2 group_3
#> Overall        381.2 87.60   89.36   99.49  106.53   56.79
#> Pretrain       396.3 87.86   88.66   93.31   96.54   73.72
#> Individual     425.2 99.07   99.54  111.68  101.85   83.67
#>
#> Support size:
#>
#> Overall    10
#> Pretrain   20 (4 common + 16 individual)
#> Individual 24

############################################################
# Alternatively -- use the C index:
############################################################
fit = ptLasso(x, y, groups, alpha = 0.5, family = "cox", type.measure = "C")
#> Warning: from glmnet C++ code (error code -30075); Numerical error at 75th
#> lambda value; solutions for larger values of lambda returned
predict(fit, xtest, groupstest, ytest = ytest)
```

```
#>
#> Call:
#> predict.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.5
#>
#> Performance (C-index):
#>
#>            allGroups   mean wtdMean group_1 group_2 group_3
#> Overall       0.8545 0.8673  0.8608  0.9139  0.7746  0.9133
#> Pretrain      0.8359 0.8396  0.8393  0.9152  0.8173  0.7864
#> Individual    0.7925 0.7985  0.8008  0.9075  0.8007  0.6873
#>
#> Support size:
#>
#> Overall    6
#> Pretrain   35 (4 common + 31 individual)
#> Individual 37
```

The call to `cv.ptLasso` is again much the same; we only need to specify `family` ("cox") and `type.measure` (if we want to use the C index instead of the partial likelihood).

```
####################################################
# Fit:
####################################################
fit = cv.ptLasso(x, y, groups, family = "cox", type.measure = "C")

####################################################
# Predict with a common alpha for all groups:
####################################################
predict(fit, xtest, groupstest, ytest = ytest)
#>
#> Call:
#> predict.cv.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest)
#>
#>
#> alpha =  0.2
#>
#> Performance (C-index):
#>
#>            allGroups   mean wtdMean group_1 group_2 group_3
#> Overall       0.8527 0.8652  0.8586  0.9113  0.7711  0.9133
#> Pretrain      0.8501 0.8795  0.8742  0.9177  0.8043  0.9164
#> Individual    0.7865 0.8005  0.8033  0.9126  0.8078  0.6811
#>
#> Support size:
#>
#> Overall    8
#> Pretrain   13 (4 common + 9 individual)
#> Individual 31
```

```r
###################################################
# Predict with a different alpha for each group:
###################################################
predict(fit, xtest, groupstest, ytest = ytest, alphatype = "varying")
#>
#> Call:
#> predict.cv.ptLasso(object = fit, xtest = xtest, groupstest = groupstest,
#>     ytest = ytest, alphatype = "varying")
#>
#>
#> alpha:
#> group_1 group_2 group_3
#>     0.3     0.4     0.4
#>
#>
#> Performance (C-index):
#>            overall   mean wtdMean group_1 group_2 group_3
#> Overall     0.8527 0.8652  0.8586  0.9113  0.7711  0.9133
#> Pretrain    0.8081 0.8493  0.8475  0.9229  0.8078  0.8173
#> Individual  0.7865 0.8005  0.8033  0.9126  0.8078  0.6811
#>
#>
#> Support size:
#>
#> Overall    8
#> Pretrain   28 (4 common + 24 individual)
#> Individual 31
```