Target grouped or multinomial reponse data

```r
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
```

Intuition

Now we turn to the **target grouped** setting, where we have a dataset with a multinomial outcome and no other grouping on the observations. For example, our data might look like the following:

```r
set.seed(1234)

n = 500; p = 75; k = 3
X = matrix(rnorm(n * p), nrow = n, ncol = p)
y = sample(1:k, n, replace = TRUE)

Xtest = matrix(rnorm(n * p), nrow = n, ncol = p)
```

Each row in X belongs to class 1, 2 or 3, and we wish to predict class membership. We could fit a single multinomial model to the data:

```r
multinomial = cv.glmnet(X, y, family = "multinomial")

multipreds  = predict(multinomial, Xtest, s = "lambda.min")
multipreds.class = apply(multipreds, 1, which.max)
```

Or, we could fit 3 one-vs-rest models; at prediction time, we would assign observations to the class with the highest probability.

```r
class1 = cv.glmnet(X, y == 1, family = "binomial")
class2 = cv.glmnet(X, y == 2, family = "binomial")
class3 = cv.glmnet(X, y == 3, family = "binomial")

ovrpreds = cbind(
  predict(class1, Xtest, s = "lambda.min"),
  predict(class2, Xtest, s = "lambda.min"),
  predict(class3, Xtest, s = "lambda.min"))
ovrpreds.class = apply(ovrpreds, 1, which.max)
```

Another alternative is to do pretraining, which fits something *in between* one model for all data and three separate models. `ptLasso` will do this for you, using the arguments `family = "multinomial"` and `use.case = "targetGroups"`.

```r
fit = ptLasso(X, y, groups = y, alpha = 0.5,
              family = "multinomial",
              use.case = "targetGroups")
```

But what exactly is pretraining doing here? We'll walk through an example, doing pretraining "by hand". The steps are:

1. Train an overall model: a multinomial model using a penalty on the coefficients β so that each coefficient is either 0 or nonzero for all classes.
2. Train individual one-vs-rest models using the penalty factor and offset defined by the overall model (as in the input grouped setting).

To train the overall model, we use `cv.glmnet` with `type.multinomial = "grouped"`. This puts a penalty on β to force coefficients to be *in* or *out* of the model for all classes. This is analogous to the overall model in the input grouped setting: we want to first learn **shared** information.

```
multinomial = cv.glmnet(X, y, family = "multinomial",
                        type.multinomial = "grouped",
                        keep = TRUE)
```

Then, we fit 3 one-vs-rest models using the support and offset from the multinomial model.

```
# The support of the overall model:
nonzero.coefs = which((coef(multinomial, s = "lambda.1se")[[1]] != 0)[-1])

# The offsets - one for each class:
offset = predict(multinomial, X, s = "lambda.1se")
offset.class1 = offset[, 1, 1]
offset.class2 = offset[, 2, 1]
offset.class3 = offset[, 3, 1]
```

Now we have everything we need to train the one-vs-rest models. As always, we have the pretraining parameter α - for this example, let's use $\alpha = 0.5$:

```
alpha = 0.5
penalty.factor = rep(1/alpha, p)
penalty.factor[nonzero.coefs] = 1

class1 = cv.glmnet(X, y == 1, family = "binomial",
                   offset = (1-alpha) * offset.class1,
                   penalty.factor = penalty.factor)
class2 = cv.glmnet(X, y == 2, family = "binomial",
                   offset = (1-alpha) * offset.class2,
                   penalty.factor = penalty.factor)
class3 = cv.glmnet(X, y == 3, family = "binomial",
                   offset = (1-alpha) * offset.class3,
                   penalty.factor = penalty.factor)
```

And we're done with pretraining! To predict, we again assign each row to the class with the highest prediction:

```
newoffset = predict(multinomial, X, s = "lambda.1se")
ovrpreds = cbind(
  predict(class1, Xtest, s = "lambda.min", newoffset = newoffset[, 1, 1]),
  predict(class2, Xtest, s = "lambda.min", newoffset = newoffset[, 2, 1]),
  predict(class3, Xtest, s = "lambda.min", newoffset = newoffset[, 3, 1])
)
ovrpreds.class = apply(ovrpreds, 1, which.max)
```

This is all done automatically within `ptLasso`; we will now show an example using the `ptLasso` functions. The example above is intended only to show how pretraining works for multinomial outcomes, and some technical details have been omitted. (For example, `ptLasso` takes care of crossfitting between the first and second steps.)

Example

First, let's simulate multinomial data with 5 classes. We start by drawing X from a normal distribution (uncorrelated features), and then we shift the columns differently for each group.

```
set.seed(1234)

n = 500; p = 50; k = 5
class.sizes = rep(n/k, k)
ncommon = 10; nindiv = 5;
shift.common = seq(-.2, .2, length.out = k)
shift.indiv  = seq(-.1, .1, length.out = k)

x     = matrix(rnorm(n * p), n, p)
xtest = matrix(rnorm(n * p), n, p)
y = ytest = c(sapply(1:length(class.sizes), function(i) rep(i, class.sizes[i])))

start = ncommon + 1
for (i in 1:k) {
  end = start + nindiv - 1
  x[y == i, 1:ncommon] = x[y == i, 1:ncommon] + shift.common[i]
  x[y == i, start:end] = x[y == i, start:end] + shift.indiv[i]

  xtest[ytest == i, 1:ncommon] = xtest[ytest == i, 1:ncommon] + shift.common[i]
  xtest[ytest == i, start:end] = xtest[ytest == i, start:end] + shift.indiv[i]
  start = end + 1
}
```

The calls to `ptLasso` and `cv.ptLasso` are almost the same as in the input grouped setting, only now we specify `use.case = "targetGroups"`. The call to `predict` does not require a `groups` argument because the groups are unknown at prediction time.

```
###############################################################################
# Fit the pretrained model.
# By default, ptLasso uses type.measure = "deviance", but for ease of
# interpretability, we use type.measure = "class" (the misclassification rate).
###############################################################################
fit = ptLasso(x = x, y = y,
              use.case = "targetGroups", type.measure = "class")

###############################################################################
# Predict
###############################################################################
predict(fit, xtest, ytest = ytest)
#>
#> Call:
#> predict.ptLasso(object = fit, xtest = xtest, ytest = ytest)
#>
#>
#>
#> alpha =  0.5
#>
#> Performance (Misclassification error):
#>
#>            overall   mean group_1 group_2 group_3 group_4 group_5
#> Overall      0.738
```

```
#> Pretrain     0.728 0.2000   0.200     0.2     0.2     0.2   0.200
#> Individual   0.736 0.1984   0.196     0.2     0.2     0.2   0.196
#>
#> Support size:
#>
#> Overall    29
#> Pretrain   23 (23 common + 0 individual)
#> Individual 32

################################################################################
# Fit with CV to choose the alpha parameter
################################################################################
cvfit = cv.ptLasso(x = x, y = y,
              use.case = "targetGroups", type.measure = "class")

################################################################################
# Predict using one alpha for all classes
################################################################################
predict(cvfit, xtest, ytest = ytest)
#>
#> Call:
#> predict.cv.ptLasso(object = cvfit, xtest = xtest, ytest = ytest)
#>
#>
#>
#> alpha =  0.9
#>
#> Performance (Misclassification error):
#>
#>             overall   mean group_1 group_2 group_3 group_4 group_5
#> Overall       0.738
#> Pretrain      0.722 0.1992     0.2     0.2     0.2     0.2   0.196
#> Individual    0.742 0.2000     0.2     0.2     0.2     0.2   0.200
#>
#> Support size:
#>
#> Overall    39
#> Pretrain   32 (23 common + 9 individual)
#> Individual 36

################################################################################
# Predict using a separate alpha for each class
################################################################################
predict(cvfit, xtest, ytest = ytest, alphatype = "varying")
#>
#> Call:
#> predict.cv.ptLasso(object = cvfit, xtest = xtest, ytest = ytest,
#>     alphatype = "varying")
#>
#>
#> alpha =  0.1 0 0.7 0 0.1
#>
#> Performance (Misclassification error):
```

```
#>
#>            overall   mean group_1 group_2 group_3 group_4 group_5
#> Overall      0.738
#> Pretrain     0.742 0.2016   0.208     0.2     0.2   0.202   0.198
#> Individual   0.742 0.2000   0.200     0.2     0.2   0.200   0.200
#>
#> Support size:
#>
#> Overall    39
#> Pretrain   36 (23 common + 13 individual)
#> Individual 36
```