Pretraining and the Lasso: a Deeper Dive

and

More Examples Using ptLasso

Erin Craig

Mert Pilanci, Thomas Le Menestrel, Balasubramanian Narasimhan,
Manuel Rivas, Roozbeh Dehghannasiri, Julia Salzman,
Jonathan Taylor, Robert Tibshirani

Background

In Part 1 of this series, we learned about **pretraining for the lasso**, as it applies to "input grouped" data: data where each observation belongs to one of k groups (e.g. cancer classes).

With simulated data, we fit pretrained lasso models in this context using the R package `ptLasso`.

In this video, we will show how to do lasso pretraining for three more settings:

1. target grouped data (data with a multinomial response),
2. multi-response data with Gaussian responses, and
3. multi-response data with time ordered responses (time series data).

A quick start

In all cases, our modeling pipeline will look like this:

```
require(ptLasso)

# Data: features X, response y,
# and "useCase" (targetGroups, multiresponse or timeSeries)
# Fit a model:
fit = ptLasso(X, y, use.case = "useCase")
# ...optionally using cross-validation for parameter selection:
fit = cv.ptLasso(X, y, use.case = "useCase")

# Visualize the model:
plot(fit)

# And make predictions:
predict(fit, Xtest)

# Inspect the coefficients
coef(fit)
```

Target grouped data
(multinomial response data)

Target grouped (multinomial response) data

- Multinomial response data is data in which responses are in one of three or more categories.
  For example, observations are measurements from irises, and we wish to predict the iris type.

Iris setosa

Iris versicolor

Iris virginica

- The pretraining philosophy for modeling data with a multinomial response is analogous to that for input grouped data: we wish to learn features that are predictive across all classes *and* to learn a specific model for each class.

Target grouped data: the method

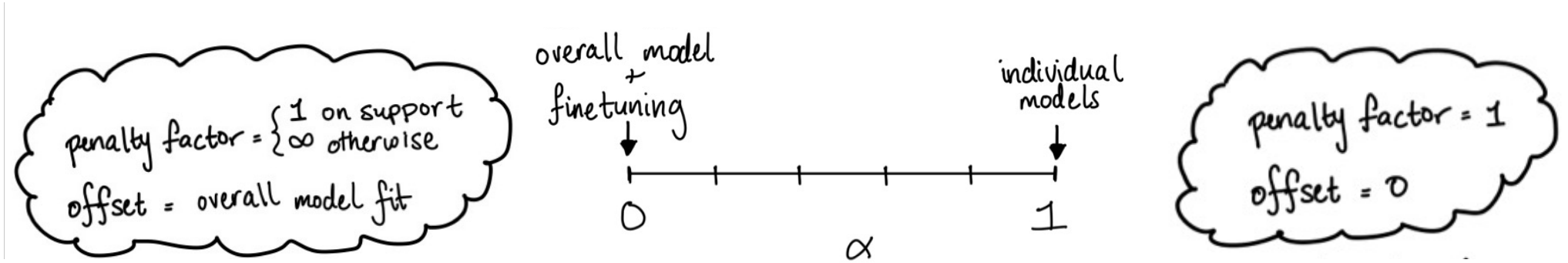
1. **Fit an overall model**:
   Fit a multinomial model using a group lasso penalty on the coefficients.
   The group lasso penalty forces all responses to use the same features,
   but allows different coefficient values.

2. **Fit class-specific models**:
   Train one-vs-rest models for each class using the penalty factor and offset defined by the overall model.
   This step allows us to discover features that are specific to each class.

Target grouped data: the method

1. **Fit an overall model**:
   Fit a multinomial model using a group lasso penalty on the coefficients.
   The group lasso penalty forces all responses to use the same features, but allows different coefficient values.

2. **Fit class-specific models**:
   Train one-vs-rest models for each class using the penalty factor and offset defined by the overall model.
   This step allows us to discover features that are predictive for each class.



Target grouped data: simulated data

We have a simulated dataset with a multinomial outcome, and we want to predict whether each observation is in group 1, 2 or 3.

There are 900 observations (300 in each group), and 100 features. There are 15 features that are predictive across all 3 classes; each class additionally has 10 class-specific features.

```
> dim(x)
[1] 900  100
```

And our response y is a vector of 1s, 2s and 3s:

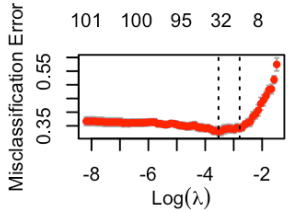
```
> head(y, 3)
[1] 1 1 1
```

(Simulation details are in the accompanying .Rmd file.)

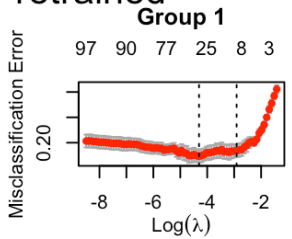
Target grouped data: model fitting

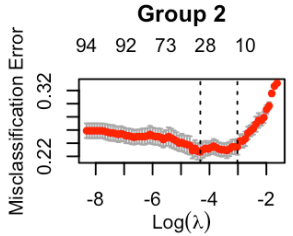
```
fit = ptLasso(x = x, y = y, use.case = "targetGroups",
              type.measure = "class", alpha = 0.5)
plot(fit)
```

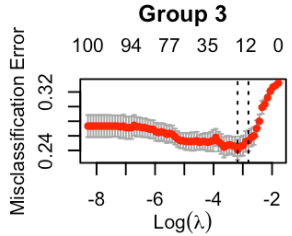
Overall



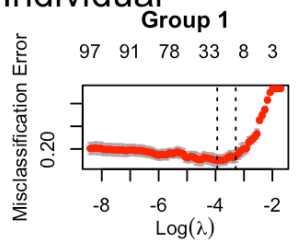
Pretrained

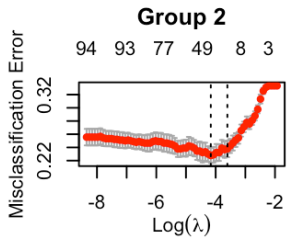


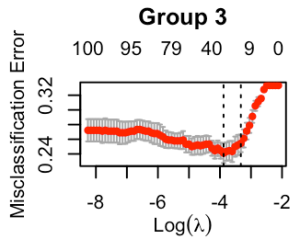




Individual







`ptLasso` used `cv.glmnet` to fit 7 models:

- 1 *overall* model (multinomial using all 3 groups),
- 3 *pretrained* models (one-vs-rest for each group),
- 3 *individual* models (one-vs-rest for each group).

`plot` shows their cross validation curves.

Target grouped data: prediction

```
predict(fit, xtest)
```

`predict` makes predictions from all 7 models. It returns a list containing:

1. `yhatoverall` (predictions from the overall model),
2. `yhatpre` (predictions from the pretrained models) and
3. `yhatind` (predictions from the individual models).

By default, `predict` uses `lambda.min` – the value of the lasso regularization parameter λ that minimized the cross-validated error rate –

for all 7 `cv.glmnet` models.

Target grouped data: prediction

```
predict(fit, xtest, ytest = ytest)

## Performance (Misclassification error):
##
##             overall   mean group_1 group_2 group_3
## Overall      0.3344
## Pretrain     0.3356 0.2289  0.1933  0.2611  0.2322
## Individual   0.3411 0.2330  0.2078  0.2578  0.2333
##
## Support size:
##
## Overall    34
## Pretrain   49 (15 common + 34 individual)
## Individual 49
```

Model interpretation

```
coefs = coef(fit)
names(coefs)
## [1] "individual" "pretrain"   "overall"
length(coefs$pretrain)
## [1] 3
head(coefs$pretrain[[3]])
## 6 x 1 sparse Matrix of class
## "dgCMatrix"
##                      s1
##(Intercept)  -0.7616493
## V1          -0.1683246
## V2           .
## V3          -0.1301920
## V4           0.2791285
## V5           .
```

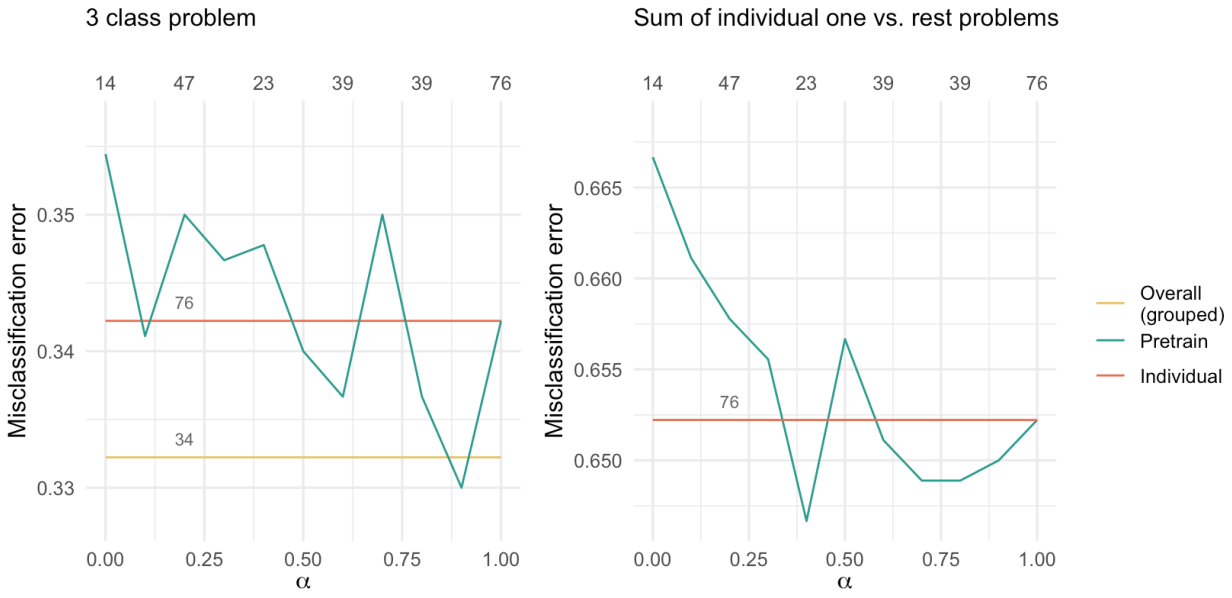
`coef` returns a list with the coefficients for the overall, individual and pretrained models.

`coefs$pretrain` is a list of length 3, containing the coefficients for each group.

Target grouped data: CV to choose α

```
cvfit = cv.ptLasso(x = x, y = y,
                   use.case = "targetGroups",
                   type.measure = "class")
```

```
plot(cvfit)
```



We can plot misclassification error as a function of α.

On the left, we visualize as a single multinomial problem; on the right we show the sum of errors from treating this as a collection of one-vs-rest problems.

Target grouped data: CV to choose α

```
predict(cvfit, xtest, ytest = ytest)
```

```
## alpha =  0.9
##
## Performance (Misclassification error):
##
##              overall   mean group_1 group_2 group_3
## Overall       0.3344
## Pretrain      0.3367 0.2341  0.2067  0.2589  0.2367
## Individual    0.3489 0.2348  0.2089  0.2611  0.2344
##
## Support size:
##
## Overall     34
## Pretrain    33 (14 common + 19 individual)
## Individual 76
```

Multi-response data

Multi-response data with Gaussian responses

- Multi-response data has *multiple* responses for each observation.
  Data like this may come from a survey, where each person answered multiple questions.
  In this case, the people would be the observations and their answers would be the responses.

- `ptLasso` supports the special case where all responses are continuous (Gaussian).

Multi-response data: the method

- Pretraining for multi-response data is similar to the multinomial setting.

- We again wish to learn across all classes *and* learn a specific model for each class. Here are the steps:
  1. **Fit an overall model**:
     Fit a multi-response model using a group lasso penalty on the coefficients.
  2. **Fit response-specific models**:
     Fit a lasso linear regression model for each response.

In step 1, we learn features that are predictive for *all* responses.
In step 2, we learn predictive features for *each* response.

We'll do an example using `ptLasso` to fit a pretrained lasso model with simulated data.

Multi-response data: simulated data

We have a simulated dataset with two Gaussian responses.

There are 650 observations and 500 features. The first 5 features are predictive for both responses; there is an extra set of 5 coefficients that are predictive for each response.

```
> dim(x)
[1] 650  500
```

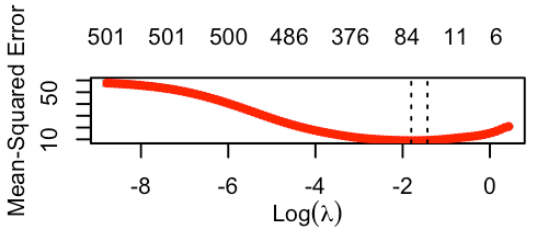
And our response y is a two-column matrix of continuous responses:

```
> head(y, 3)
          [,1]       [,2]
[1,] -6.820289 -3.1130501
[2,]  2.017584  4.3513815
[3,] -4.069882 -0.8736443
```

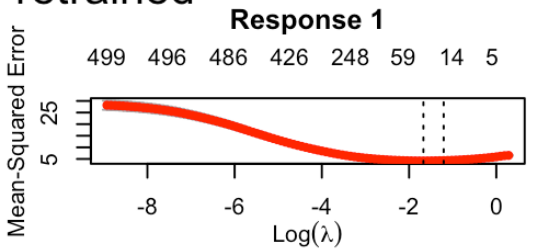
Multi-response data: model fitting

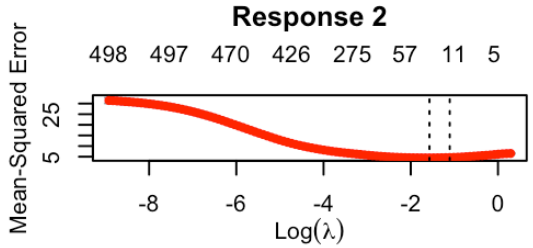
```
fit = ptLasso(x = x, y = y, use.case = "multiresponse", alpha = 0.5)
plot(fit)
```

Overall

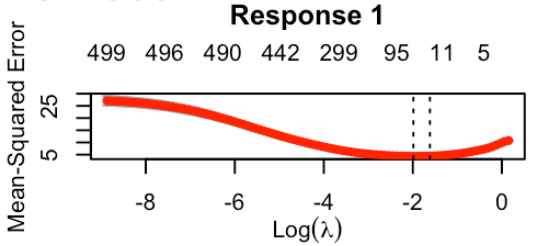


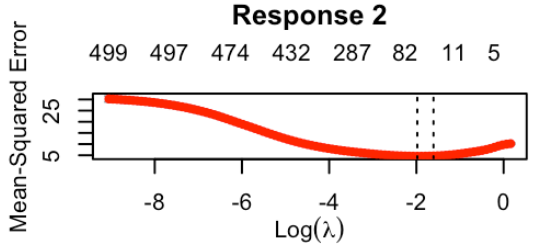
Pretrained





Individual





`ptLasso` used `cv.glmnet` to fit 5 models:

- 1 *overall* model (multi-response),
- 2 *pretrained* models (one for each response),
- 2 *individual* models (one for each response).

`plot` shows their cross validation curves.

Multi-response data: prediction

```
predict(fit, xtest)
```

`predict` makes predictions from all 5 models. It returns a list containing:

1. `yhatoverall` (predictions from the overall model),
2. `yhatpre` (predictions from the pretrained models) and
3. `yhatind` (predictions from the individual models).

Multi-response data: prediction

```
predict(fit, xtest, ytest = ytest)

## Performance (Mean squared error):
##
##              overall mean  response_1 response _2
## Overall       9.217  4.608      4.092          5.125
## Pretrain      9.072  4.536      4.132          4.940
## Individual    9.324  4.662      4.168          5.157
##
## Support size:
##
## Overall     57
## Pretrain    23 (20 common + 3 individual)
## Individual 75
```

Model interpretation

```
coefs = coef(fit)
names(coefs)
## [1] "individual" "pretrain"   "overall"
length(coefs$pretrain)
## [1] 2
head(coefs$pretrain[[2]])
## 6 x 1 sparse Matrix of class
## "dgCMatrix"
##                      s1
## (Intercept) -0.03083231
## V1           0.40342178
## V2           0.35554835
## V3           0.45478266
## V4           0.43759794
## V5           0.39706698
```

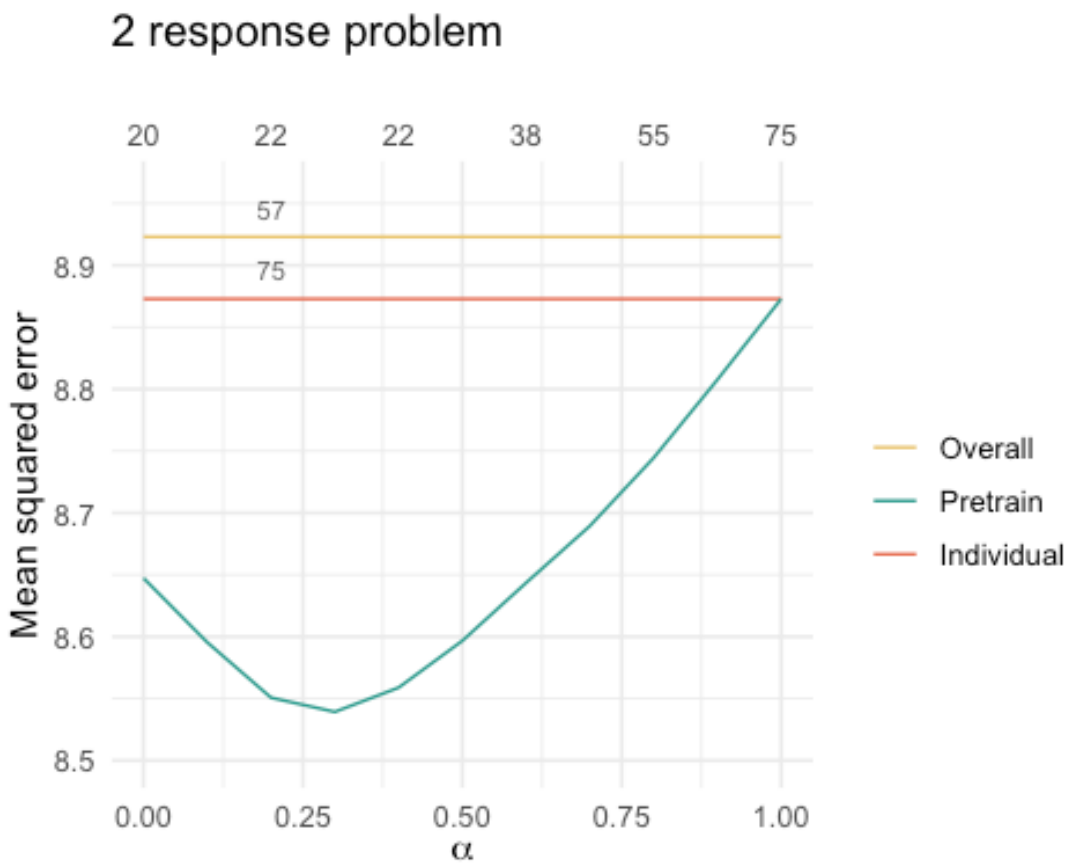
`coef` returns a list with the coefficients for the overall, individual and pretrained models.

`coefs$pretrain` is a list of length 2, containing the coefficients for each response.

Multi-response data: CV to choose α

```
cvfit = cv.ptLasso(x = x, y = y, use.case = "multiresponse")

plot(cvfit)
```



We can plot cross validated mean squared error as a function of α.

This plot shows the sum of mean squared errors across the two responses.

Multi-response data: CV to choose α

```
predict(cvfit, xtest, ytest = ytest)

## alpha = 0.3
##
## Performance (Mean squared error):
##
##             allResponses  mean response_1 response_2
## Overall            9.217 4.608      4.092      5.125
## Pretrain           9.006 4.503      4.149      4.857
## Individual         9.324 4.662      4.168      5.157
##
## Support size:
##
## Overall    57
## Pretrain   22 (20 common + 2 individual)
## Individual 75
```

Time ordered multi-response data
(time series data)

Time ordered multi-response data

- Here, we again have multiple responses for each observation.

- Now, however, these responses are measurements of the *same outcome at different moments in time*. For example, we may observe patients across many hospital visits, and measure the same biomarker at each visit. In this case, the time ordered biomarker measurements would be the responses.

- `ptLasso` has built-in support for data like this – we'll call it time series data – where all responses are either Gaussian or binomial.

- `ptLasso` also supports the setting where the *features* are measured separately at each time point.

Time series data: the method

Our goal is to leverage signal across time points.

To do pretraining with time series data, we:

1. **Fit an initial model**:
   Fit a lasso penalized model for the first time point.
2. **Fit the next time-specific model**:
   Train a model for time 2 using the offset and penalty factor defined by time 1.
3. **Continue**:
   Train a model for time 3 using the offset from time 2 and the union of supports from times 1 and 2.
   Train a model for time 4 using the offset from time 3 and the union of supports from times 1, 2 and 3.
   etc.

Time series data: simulated data

We have a simulated dataset with two Gaussian responses.

There are 300 observations and 100 features. The first 10 features are predictive for both responses; the second response has 5 additional predictive features.

```
> dim(x)
[1] 300  100
```

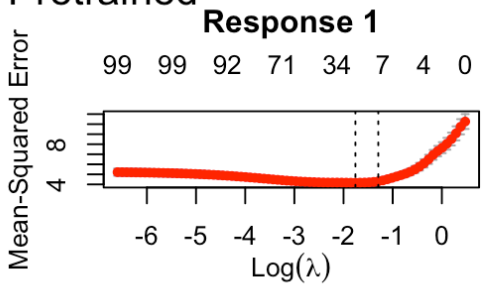
And our response y is a two-column matrix of continuous responses:

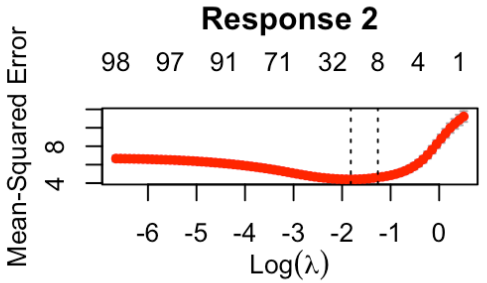
```
> head(y, 3)
          [,1]       [,2]
[1,]  4.7966632  0.2754762
[2,] -0.7150173 -2.7710819
[3,] -2.4339730 -0.2602280
```

Time series data: model fitting

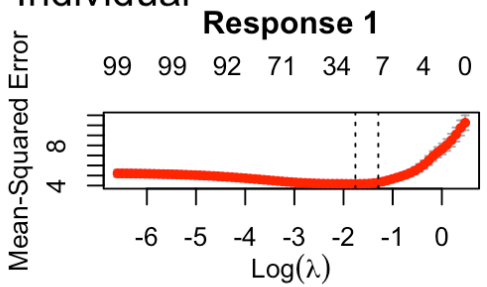
```
fit = ptLasso(x = x, y = y, use.case = "timeseries", alpha = 0.5)
plot(fit)
```

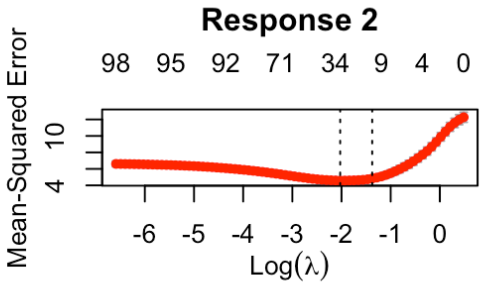
Pretrained





Individual





`ptLasso` used `cv.glmnet` to fit 4 models:

- 2 *pretrained* models (one for each response),
- 2 *individual* models (one for each response).

`plot` shows their cross validation curves.

Time series data: prediction

```
predict(fit, xtest)
```

`predict` makes predictions from all 4 models. It returns a list containing:

1. `yhatpre` (predictions from the pretrained models) and
2. `yhatind` (predictions from the individual models).

Time series data: prediction

```
predict(fit, xtest, ytest = ytest)

## Performance (Mean squared error):
##
##              mean  response_1 response _2
## Pretrain    4.664       4.609        4.719
## Individual 4.454        4.609        4.299
##
## Support size:
##
## Pretrain    34 (6 common + 28 individual)
## Individual 39
```

Model interpretation

```
coefs = coef(fit)
names(coefs)
## [1] "individual" "pretrain"
length(coefs$pretrain)
## [1] 2
head(coefs$pretrain[[2]])
## 6 x 1 sparse Matrix of class
## "dgCMatrix"
##                       s1
## (Intercept)  0.05059930
## V1                    .
## V2                    .
## V3                    .
## V4                    .
## V5           -0.02437197
```

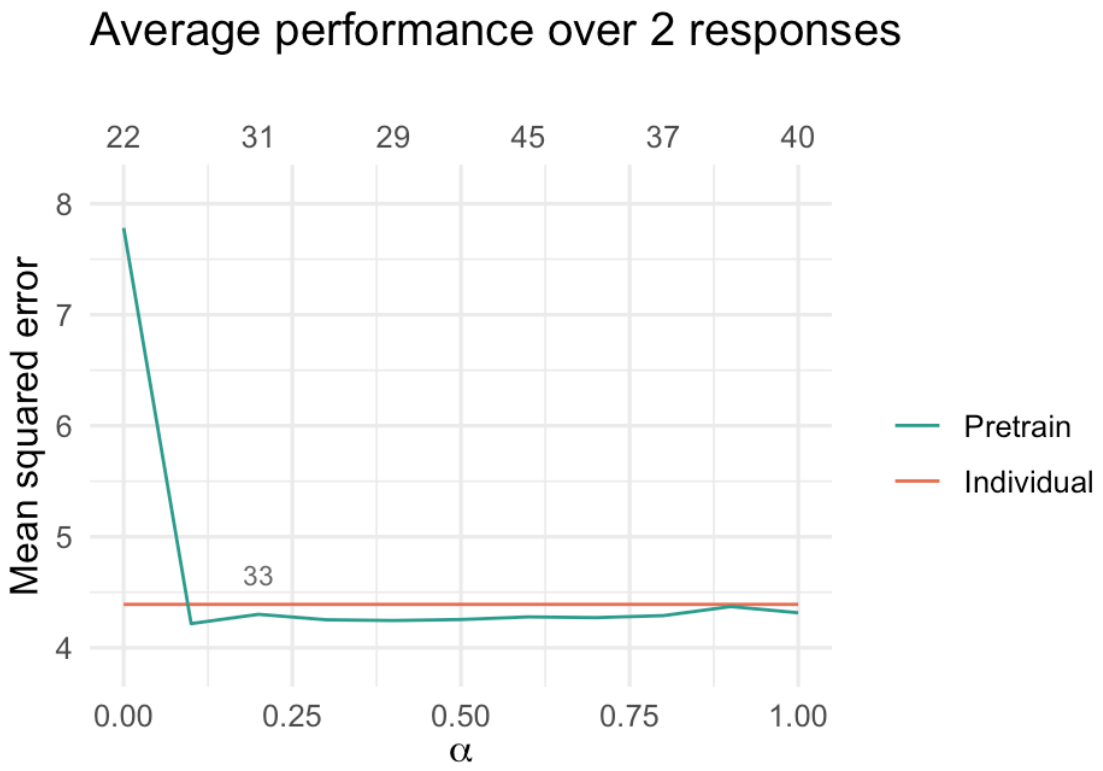
`coef` returns a list with the coefficients for the individual and pretrained models.

`coefs$pretrain` is a list of length 2, containing the coefficients for both responses.

Time series data: CV to choose α

```
cvfit = cv.ptLasso(x = x, y = y, use.case = "timeseries")

plot(cvfit)
```



We can plot cross validated mean squared error as a function of α.

This plot shows the average mean squared error across the two responses.

Time series data: CV to choose α

```
predict(cvfit, xtest, ytest = ytest)

## alpha = 0.1
## Performance (Mean squared error):
##
##                mean  response_1 response_2
## Pretrain      4.343       4.573       4.112
## Individual    4.502       3.573       4.431
##
## Support size:
##
## Pretrain    37 (8 common + 29 individual)
## Individual 33
```

Pretraining with time series data: an extension

In addition to repeated measurements of *y* across time, we may also have repeated measurements of our features *X* at each point in time.

`ptLasso` supports this setting: simply pass in a *list* for *X* with one entry for each time point.

To see an example, please look at the `ptLasso` vignette.

Wrap up

Thank you for watching!

Now we know how to do pretraining for all of the settings covered by `ptLasso`:

- Input groups
- Target groups (multinomial response)
- Multi-response Gaussian
- Time-ordered Gaussian or binomial

If you want to see more example use cases of pretraining –
including conditional average treatment effect estimation and
multi-response data with different outcome types –
check out the `ptLasso` vignette.