Introduction

1 Introduction to pretraining

Suppose we have a dataset spanning ten cancers and we want to fit a lasso penalized Cox model to predict survival time. Some of the cancer classes in our dataset are large (e.g. breast, lung) and some are small (e.g. head and neck). There are two obvious approaches: (1) fit a "pancancer model" to the entire training set and use it to make predictions for all cancer classes and (2) fit a separate (class specific) model for each cancer and use it to make predictions for that class only. Pretraining is a method that bridges these two options; it has a hyperparameter that allows you to fit the pancancer model, the class specific models, and everything in between.

`ptLasso` is a package that fits pretrained models using the `glmnet` package (Tay, Narasimhan, and Hastie (2023)), including lasso, elasticnet and ridge models .

Our example dataset consisting of ten different cancers is called **input grouped**. There is a grouping on the rows of X and each row belongs to one of the cancer classes. Alternatively, data can be **target grouped**, where there is no grouping on the rows of X, but we have (for example) a multinomial outcome. We could fit one multinomial model, or we could fit a set of one-vs-rest models. Pretraining again bridges the two approaches, and this is described in detail in the section "Target grouped data". The remainder of this introduction describes the input grouped setting.

Pretraining is a general method to pass information from one model to another – it has many uses beyond what has already been discussed here, including time series data, multi-response data with mixed response types, and multitask learning. Some of these modeling tasks are not supported by the `ptLasso` package, and this vignette shows how to do pretraining for them using the `glmnet` package.

Before we describe pretraining in more detail, we will first give a quick review of the lasso.

1.1 Review of the lasso

For the Gaussian family with data $(x_i, y_i), i = 1, 2, \dots n$, the lasso has the form

$$\text{argmin}_{\beta_0,\beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|. \tag{1}$$

Varying the regularization parameter $\lambda \geq 0$ yields a path of solutions: an optimal value $\hat{\lambda}$ is usually chosen by cross-validation, using for example the `cv.glmnet` function from the package `glmnet`.

In GLMs and ℓ_1-~regularized GLMs, one can include an *offset*: a pre-specified n-vector that is included as an additional column to the feature matrix, but whose weight β_j is fixed at 1. Secondly, one can generalize the ℓ_1 norm to a weighted norm, taking the form

$$\sum_j \text{pf}_j |\beta_j| \tag{2}$$

where each $\text{pf}_j \geq 0$ is a **penalty factor** for feature j. At the extremes, a penalty factor of zero implies no penalty and means that the feature will always be included in the model; a penalty factor of $+\infty$ leads to that feature being discarded (i.e., never entered into the model).

1.2 Details of pretraining

Pretraining model fitting happens in two steps. First, train a model using the full data:

$$\hat{\mu}_0, \hat{\theta}_2, \dots, \hat{\theta}_K, \hat{\beta}_0 = \arg \min_{\mu, \theta_1, \dots, \theta_{k-1}, \beta} \frac{1}{2} \sum_{k=1}^{K} \| y_k - (\mu \mathbf{1} + \theta_k \mathbf{1} + X_k \beta) \|_2^2 + \lambda ||\beta||_1, \tag{3}$$

where:

- X_k, y_k are the observations in group k,
- θ_k is the group specific intercept for group k (by convention, $\hat{\theta}_1 = 0$),
- μ, β are the overall intercept and coefficients,
- and λ is a hyperparameter that has been chosen (perhaps the value minimizing the CV error).

Define $S(\hat{\beta}_0)$ to be the support set (the nonzero coefficients) of $\hat{\beta}_0$.

Then, for each group k, fit an *individual* model: find $\hat{\beta}_k$ and $\hat{\mu}_k$ such that

$$\begin{aligned}\hat{\mu}_k, \hat{\beta}_k = \arg \min_{\mu, \beta} \frac{1}{2} \| y_k - (1 - \alpha) \left(\hat{\mu}_0 \mathbf{1} + \hat{\theta}_k \mathbf{1} + X_k \hat{\beta}_0 \right) - (\mu \mathbf{1} + X_k \beta) \|_2^2 + \\ \lambda \sum_{j=1}^{p} \Big[I(j \in S(\hat{\beta}_0)) + \frac{1}{\alpha} I(j \notin S(\hat{\beta}_0)) \Big] |\beta_j|,\end{aligned} \tag{4}$$

where $\lambda > 0$ and $\alpha \in [0, 1]$ are hyperparameters that may be chosen through cross validation.

Note that this is a lasso linear regression model with *offset* $(1 - \alpha) \left(\hat{\mu}_0 \mathbf{1} + \hat{\theta}_k \mathbf{1} + X_k \hat{\beta}_0 \right)$ and coefficient j has *penalty factor* 1 if $j \in S(\hat{\beta}_0)$ and $\frac{1}{\alpha}$ otherwise.

Notice that when $\alpha = 0$, this returns the overall model fine tuned for each group: this second stage model is only allowed to fit the residual $y_k - \left(\hat{\mu}_0 \mathbf{1} + \hat{\theta}_k \mathbf{1} + X_k \hat{\beta}_0 \right)$, and the penalty factor $I(j \in S(\hat{\beta}_0)) + \infty I(j \notin S(\hat{\beta}_0))$ disallows the use of β_j unless it was already selected by the overall model.

At the other extreme, when $\alpha = 1$, this is equivalent to fitting a separate model for each class. There is no offset, and the lasso penalty is 1 for all features (the usual lasso penalty).

1.3 `ptLasso` under the hood

All model fitting in `ptLasso` is done with `cv.glmnet`. The first step of pretraining is a straightforward call to `cv.glmnet`; the second step is done by calling `cv.glmnet` with:

1.`offset` $(1 - \alpha) \left(\hat{\mu}_0 \mathbf{1} + X_k \hat{\beta}_0 \right)$ and 2.`penalty.factor`, the j^{th} entry of which is 1 if $j \in S(\hat{\beta}_0)$ and $\frac{1}{\alpha}$ otherwise.

Because `ptLasso` uses `cv.glmnet`, it inherits most of the virtues of the `glmnet` package: for example, it handles sparse input-matrix formats, as well as range constraints on coefficients.

Additionally, one call to `ptLasso` fits an overall model, pretrained class specific models, and class specific models for each group (without pretraining). The `ptLasso` package also includes methods for prediction and plotting, and a function that performs K-fold cross-validation.

2 Installation

To install this package, do the following.

```
require(remotes)
remotes::install_github("erincr/ptLasso")
```

Table 1: Coefficients for simulating input grouped data

	1-10	11-20	21-30	31-40	41-59	51-60	61-120
group 1	3	3	0	0	0	0	0
group 2	6	0	3	0	0	0	0
group 3	9	0	0	3	0	0	0
group 4	12	0	0	0	3	0	0
group 5	15	0	0	0	0	3	0

3 Quick start

This section shows how to use the main functions in `ptLasso`. We will show more details and options in the following sections. First, we load the `ptLasso` package:

```
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
```

To show how to use `ptLasso`, we'll simulate data with 5 groups and a continuous response using the helper function `gaussian.example.data`. There are $n = 200$ observations in each group and $p = 120$ features. All groups share 10 informative features; though the features are shared, they have different coefficient values. Each group has 10 additional features that are specific to that group, and all other features are uninformative.

The coefficients for the 5 groups are in Table 1.

```
## Loading required package: knitr
```

```
set.seed(1234)

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

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

Now we are ready to fit a model using `ptLasso`. We'll start by defining the pretraining hyperparamter $\alpha = 0.5$ (randomly chosen). In practice we recommend using a validation set to measure performance for a few different choices of α, or using `cv.ptLasso`, which will recommend a choice of α based on CV performance.

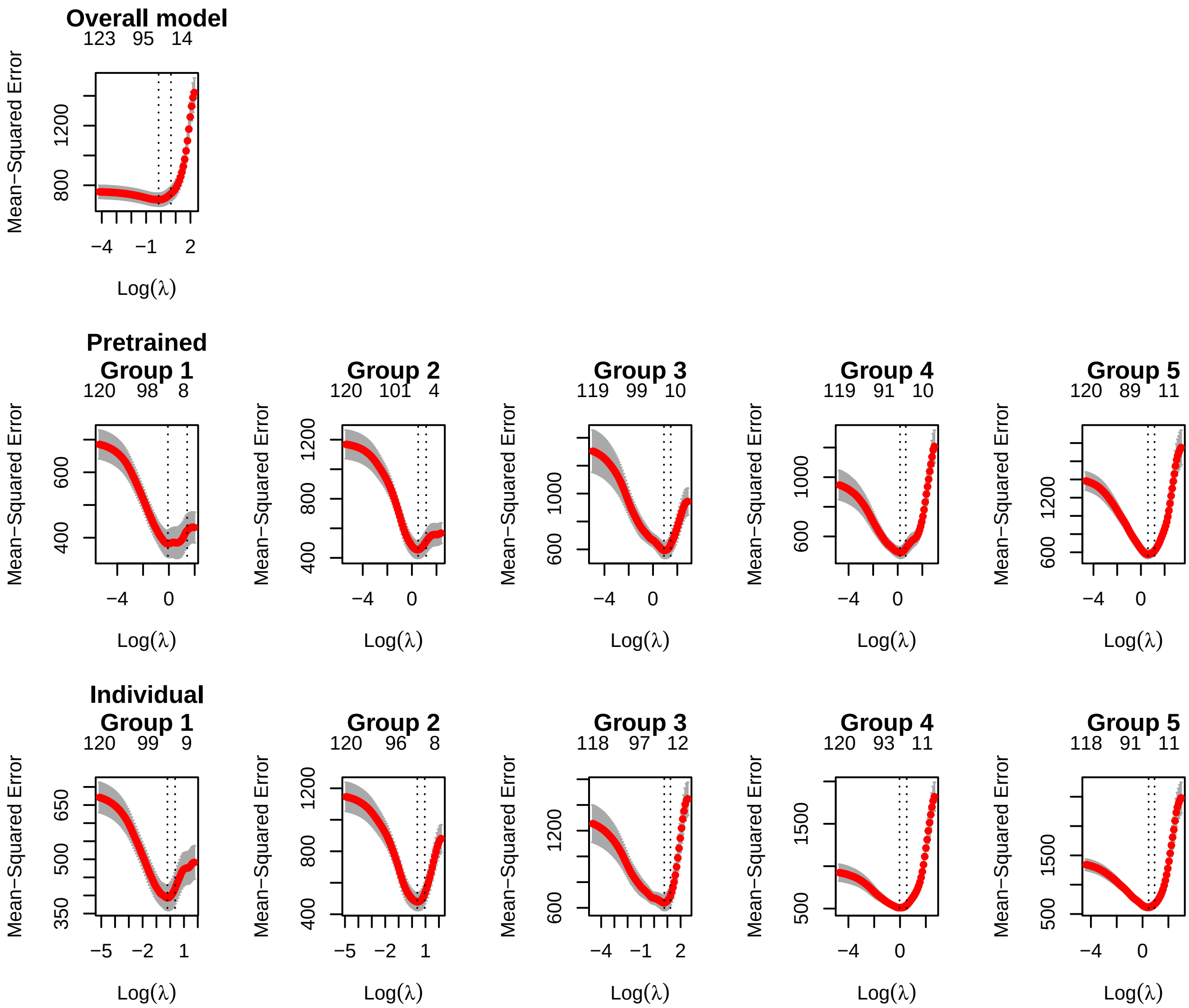
```
fit <- ptLasso(x, y, groups, alpha = 0.5)
```

The function `ptLasso` used `cv.glmnet` to fit 11 models:

- the *overall* model (using all 5 groups),
- the 5 *pretrained* models (one for each group) and
- the 5 *individual* models (one for each group).

A call to `plot` will show us the cross validation curves for each model. The top row shows the overall model, the middle row the pretrained models, and the bottom row the individual models.

```r
plot(fit)
```



`predict` makes predictions from all 11 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` for all 11 `cv.glmnet` models; you could instead specify `s = lambda.1se` or use a numeric value. Whatever value of λ you choose will be used for all models (overall, pretrained and individual).

```r
preds = predict(fit, xtest, groupstest=groupstest)
```

If you also provide `ytest` (e.g. for model validation), `predict` will additionally compute performance measures.

```r
preds = predict(fit, xtest, groupstest=groupstest, ytest=ytest)
preds
```

```
##
## Call:
## predict.ptLasso(fit = fit, xtest = xtest, groupstest = groupstest,
```

```
##     ytest = ytest)
##
##
## alpha =  0.5
##
## Performance (Mean squared error):
##
##            allGroups  mean group_1 group_2 group_3 group_4 group_5    r^2
## Overall        758.6 758.6   805.1   534.9   568.7   802.6  1081.5 0.5353
## Pretrain       493.8 493.8   550.9   428.7   518.8   496.7   473.9 0.6975
## Individual     532.8 532.8   584.1   443.2   567.2   550.5   518.9 0.6736
##
## Support size:
##
## Overall    47
## Pretrain   98 (16 common + 82 individual)
## Individual 109
```

To access the coefficients of the fitted models, use `coef` as usual. This returns a list with the coefficients of the individual models, pretrained models and overall models, as returned by `glmnet`.

```r
all.coefs = coef(fit, s= "lambda.min")
names(all.coefs)
```

```
## [1] "individual" "pretrain"   "overall"
```

The entries for the individual and pretrained models are lists with one entry for each group. Because we have 5 groups, we'll have 5 sets of coefficients.

```r
length(all.coefs$pretrain)
```

```
## [1] 5
```

The first few coefficients for group 1 from the pretrained model are:

```r
head(all.coefs$pretrain[[1]])
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept)  0.44678793
## V1          -3.96783783
## V2           .
## V3           .
## V4          -0.09154089
## V5          -0.85125296
```

When we used `ptLasso` to fit a model, we chose $\alpha = 0.5$. If we want to use cross validation to compare many choices of α, we can use `cv.ptLasso`. After fitting, the `cv.ptLasso` object will print out the cross validated mean squared error for (1) the overall model, (2) the pretrained models for all compared choices of α and (3) the individual models.

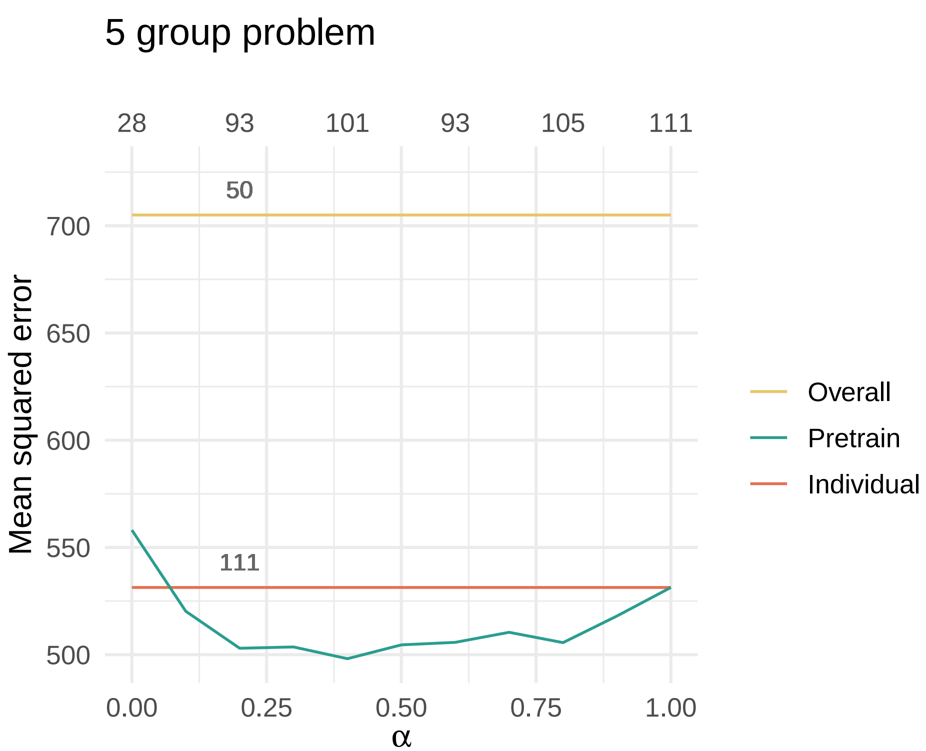
```r
cvfit <- cv.ptLasso(x, y, groups)
cvfit
```

```
##
## Call:
## cv.ptLasso(x = x, y = y, groups = groups, family = "gaussian",
##     type.measure = "mse", use.case = "inputGroups")
##
```

```
##
##
## type.measure:  mse
##
##
##             alpha overall  mean wtdMean group_1 group_2 group_3 group_4 group_5
## Overall             705.0 705.0   705.0   735.3   519.2   566.4   667.2  1037.0
## Pretrain      0.0   558.1 558.1   558.1   474.8   516.3   607.1   563.4   629.0
## Pretrain      0.1   520.2 520.2   520.2   417.0   470.7   620.7   511.4   581.4
## Pretrain      0.2   503.0 503.0   503.0   410.1   464.6   608.2   486.1   545.9
## Pretrain      0.3   503.6 503.6   503.6   427.1   478.2   571.2   479.6   561.8
## Pretrain      0.4   498.1 498.1   498.1   377.1   464.1   582.7   496.3   570.4
## Pretrain      0.5   504.6 504.6   504.6   376.9   478.6   590.5   500.7   576.3
## Pretrain      0.6   505.7 505.7   505.7   382.9   467.0   616.8   493.8   568.2
## Pretrain      0.7   510.4 510.4   510.4   398.9   482.5   603.3   471.3   596.1
## Pretrain      0.8   505.6 505.6   505.6   378.5   483.7   593.7   502.6   569.5
## Pretrain      0.9   518.0 518.0   518.0   419.3   485.9   596.8   518.3   569.8
## Pretrain      1.0   531.3 531.3   531.3   416.2   509.3   613.9   509.4   607.9
## Individual          531.3 531.3   531.3   416.2   509.3   613.9   509.4   607.9
##
## alphahat (fixed) = 0.4
## alphahat (varying):
## group_1 group_2 group_3 group_4 group_5
##     0.5     0.4     0.3     0.7     0.2
```

We can plot the `cv.ptLasso` object to visualize performance as a function of α:

```r
plot(cvfit)
```



And, as with `ptLasso`, we can `predict`. By default, `predict` uses the α that minimized the cross validated MSE:

```r
preds = predict(cvfit, xtest, groupstest=groupstest, ytest=ytest)
preds
```

```
##
## Call:
## predict.cv.ptLasso(cvfit = cvfit, xtest = xtest, groupstest = groupstest,
##     ytest = ytest)
##
##
## alpha =  0.4
##
## Performance (Mean squared error):
##
##            allGroups  mean group_1 group_2 group_3 group_4 group_5    r^2
## Overall        757.1 757.1   815.7   542.6   567.1   792.7  1067.5 0.5362
## Pretrain       501.9 501.9   585.7   439.0   519.4   494.7   470.6 0.6926
## Individual     529.3 529.3   572.6   441.8   562.4   550.5   518.9 0.6758
##
## Support size:
##
## Overall    50
## Pretrain   101 (20 common + 81 individual)
## Individual 111
```

We could instead use the argument `alphatype = "varying"` to use the α that minimizes the CV MSE for each individual group:

```r
preds = predict(cvfit, xtest, groupstest=groupstest, ytest=ytest,
                alphatype="varying")
preds
```

```
##
## Call:
## predict.cv.ptLasso(cvfit = cvfit, xtest = xtest, groupstest = groupstest,
##     ytest = ytest, alphatype = "varying")
##
##
## alpha:
## group_1 group_2 group_3 group_4 group_5
##     0.5     0.4     0.3     0.7     0.2
##
##
## Performance (Mean squared error):
##            overall  mean wtdMean group_1 group_2 group_3 group_4 group_5
## Overall      757.1 757.1   757.1   815.7   542.6   567.1   792.7  1067.5
## Pretrain     485.3 485.3   485.3   490.7   439.0   517.1   520.3   459.5
## Individual   529.3 529.3   529.3   572.6   441.8   562.4   550.5   518.9
##
##
## Support size:
##
## Overall    50
## Pretrain   94 (20 common + 74 individual)
## Individual 111
```

Tay, J. Kenneth, Balasubramanian Narasimhan, and Trevor Hastie. 2023. "Elastic Net Regularization Paths for All Generalized Linear Models." *Journal of Statistical Software* 106 (1): 1–31. https://doi.org/10.18637/jss.v106.i01.