Time series data

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

We may have repeated measurements of X and y across time; for example, we may observe patients at two different points in time. We expect that the relationship between X and y will be different at time 1 and time 2, but not completely unrelated. Therefore, pretraining can be useful: we can use the model fitted at time 1 to inform the model for time 2.

`ptLasso` does not natively support this setting, but we can use pretraining nonetheless – below is an example. We assume that X has changed between times 1 and 2. However, if X is constant across time, we can also treat this as a multitask problem – see the section "Multitask learning or coaching" for an example.

To do pretraining, our plan is as follows:

1. fit a model for time 1 and extract its offset and support,
2. use the offset and support (the usual pretraining) to train a model for time 2.

We'll start by simulating data – more details in the comments.

```r
set.seed(1234)

n = 600; ntrain = 300;
p = 20

# We assume that X at time 1 (x1) and X at time 2 (x2) are related:
# to get X2, we modify X1.
x1 = matrix(rnorm(n*p), n, p)
x2 = x1 + matrix(0.2 * rnorm(n*p), n, p)

# The relationship between X and y at time 1 and 2 will be similarly related.
# The coefficients at time 2 are a function of those at time 1.
# Importantly, they share the same support.
beta1 = c(rep(2, 10), rep(0, p-10))
beta2 = runif(p, 0.5, 1)*beta1

# Finally, we compute y.
# y2 is a function of y1, x2 and beta2.
y1 = x1 %*% beta1 + rnorm(n)
y2 = 0.5 * y1 + x2 %*% beta2 + rnorm(n)
```

Split into train and test, and define folds to use for cross validation:

```r
# Split into train and test:
x1test = x1[-(1:ntrain), ]
x2test = x2[-(1:ntrain), ]
y1test = y1[-(1:ntrain)]
y2test = y2[-(1:ntrain)]

x1 = x1[1:ntrain, ]
x2 = x2[1:ntrain, ]
```

```
y1 = y1[1:ntrain]
y2 = y2[1:ntrain]

# Define 10 training folds:
nfolds = 10
foldid = sample(rep(1:10, trunc(nrow(x1)/nfolds)+1))[1:nrow(x1)]
```

Now we have our simulated data and we are ready to train models. The first step is to fit a model for time 1 and extract the cross-fitted offset and support. Note that `cv.glmnet` will store cross-fitted predictions if we use the argument `keep = TRUE`.

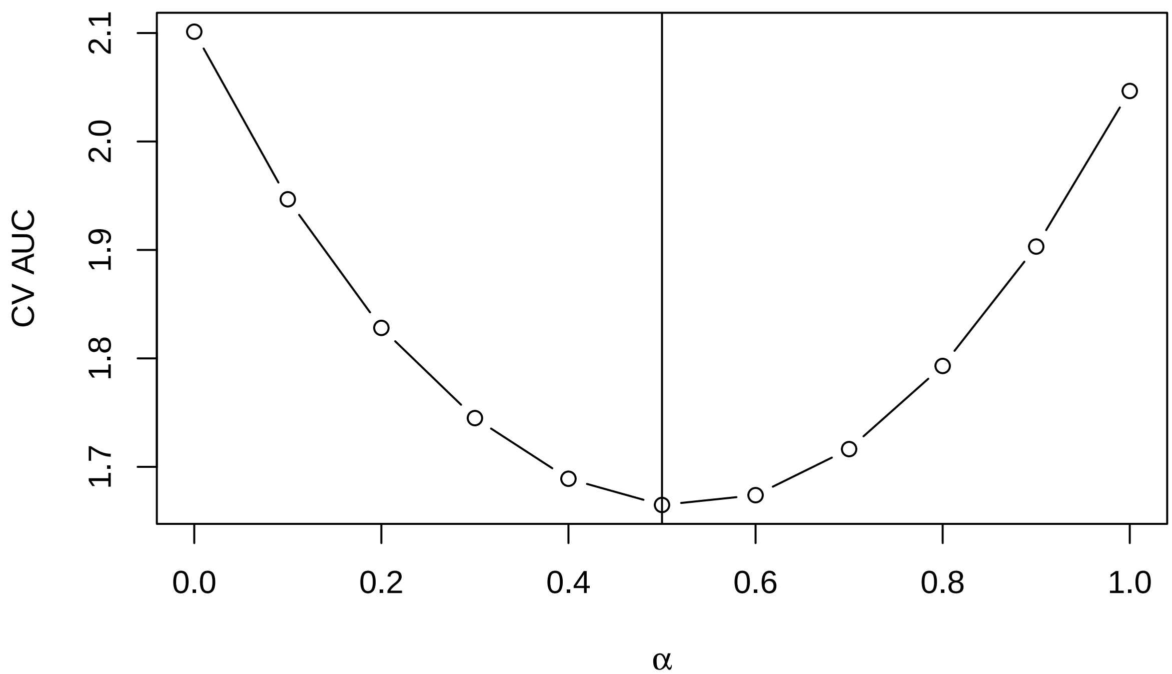
```
y1_fit = cv.glmnet(x1, y1, keep=TRUE, foldid = foldid)

# Identify the support: coefficients which are nonzero:
support = which(coef(y1_fit, s = y1_fit$lambda.1se)[-1] != 0)

# Glmnet computed the cross-fitted predictions:
offset = y1_fit$fit.preval[, y1_fit$lambda == y1_fit$lambda.1se]
```

The last step is to train a model for time 2 using the offset and support from the previous model. As always with pretraining, there is a hyperparameter α that determines the influence of the time 1 model on the time 2 model; we can choose this with cross validation. Here, we train models for a range of values of α (0, 0.1, 0.2, … 1), and store the cross validated MSE – we will choose α corresponding to the model with the lowest CV MSE.

```
cv.error = NULL
alphalist = seq(0, 1, length.out = 11)

for(alpha in alphalist){
  # Penalty factor:
  pf = rep(1/alpha, p)
  pf[support] = 1

  # Offset:
  offset.alpha = (1 - alpha) * offset

  # Model fitting:
  y2_fit = cv.glmnet(x2, y2,
                     foldid = foldid,
                     offset = offset.alpha,
                     penalty.factor = pf)

  # Use the CV MSE computed by cv.glmnet:
  cv.error = c(cv.error, min(y2_fit$cvm))
}
```

Which α gave us the best performance? Plotting the CV MSE across all αs we compared reveals that the best $\alpha = 0.5$.

```
plot(alphalist, cv.error, type = "b",
     xlab = expression(alpha),
     ylab = "CV AUC")
best.alpha = alphalist[which.min(cv.error)]
abline(v = alphalist[which.min(cv.error)])
```




```
# Train a model using alpha = 0.5
pf = rep(1/best.alpha, p)
pf[support] = 1

offset.alpha = (1-best.alpha) * offset

y2_fit = cv.glmnet(x2, y2, foldid = foldid,
                   offset = offset.alpha,
                   penalty.factor = pf)
```

Out of curiosity, let's train an entirely separate model for time 2 (though we have done this already – this is the special case of pretraining where $\alpha = 1$). This will give us a baseline performance measure.

```
y2_fit_no_pretrain = cv.glmnet(x2, y2, foldid = foldid)

testoffset = (1 - best.alpha) * predict(y1_fit, x1test, s="lambda.1se")
pretrain_preds = predict(y2_fit, x2test, newoffset = testoffset)
cat("Pretrain PSE:", round(mean((y2test - pretrain_preds)^2), 2), "\n")
#> Pretrain PSE: 1.35

individual_preds = predict(y2_fit_no_pretrain, x2test)
cat("Individual PSE:", round(mean((y2test - individual_preds)^2), 2))
#> Individual PSE: 1.7
```

Pretraining gives us a 20% lower PSE than just using individual models. This is not surprising – we simulated data to favor pretraining. Recall, however, that if the true models at times 1 and 2 are unrelated, cross validation over the pretraining hyperparameter α will encourage us to choose the individual model, and pretraining should not hurt our performance.