Different groups in train and test data

Suppose we observe groups at test time that were unobserved at train time. For example, our training set may consist of k *people* – each with many observations – and at test time, we wish to make predictions for observations from new people.

We can still use pretraining in this setting. Now however, we also fit an extra model to predict the similarity of test observations to those from the *training people*: for each observation, this model gives us a similarity (or probability) vector of length k that sums to 1. Then, we make predictions from each pretrained person-specific model and compute the weighted average prediction with respect to the similarity vector.

Here, we show an example using simulated data.

```
#> Loading required package: glmnet
#> Loading required package: Matrix
#> Loaded glmnet 4.1-8
#> Loading required package: ptLasso
#> Loading required package: ggplot2
#> Loading required package: gridExtra
```

```
set.seed(1234)

# Start with 5 people, each with 300 observations and 200 features.
# 3 people will be used for training, and 2 for testing.
n = 300*5; p = 200;
groups = sort(rep(1:5, n/5))

# We will have different coefficients for each of the 3 training people,
# and the first 3 features are shared support.
beta.group1 = c(-1, 1, 1, rep(1, 3), rep(0, p-6));
beta.group2 = c(-1, 1, 1, rep(0, 3), rep(1, 3), rep(0, p-9));
beta.group3 = c(-1, 1, 1, rep(0, 6), rep(1, 3), rep(0, p-12));

# The two test people are each a combination of of the training people.
# Person 4 will have observations drawn from classes 1 and 2, and
# Person 5 will have observations drawn from classes 1 and 3.
# The vector "hidden groups" is a latent variable - used to simulate data
# but unobserved in real data.
hidden.gps = groups
hidden.gps[hidden.gps == 4] = sample(c(1, 2), sum(groups == 4), replace = TRUE)
hidden.gps[hidden.gps == 5] = sample(c(1, 3), sum(groups == 5), replace = TRUE)

# We modify X according to group membership;
# we want X to cluster into groups 1, 2 and 3.
x = matrix(rnorm(n * p), nrow = n, ncol = p)
x[hidden.gps == 1, 1:3] = x[hidden.gps == 1, 1:3] + 1
x[hidden.gps == 2, 1:3] = x[hidden.gps == 2, 1:3] + 2
x[hidden.gps == 3, 1:3] = x[hidden.gps == 3, 1:3] + 3

# And now, we compute y using betas 1, 2 and 3:
```

```r
x.beta = rep(0, n)
x.beta[hidden.gps == 1] = x[hidden.gps == 1, ] %*% beta.group1
x.beta[hidden.gps == 2] = x[hidden.gps == 2, ] %*% beta.group2
x.beta[hidden.gps == 3] = x[hidden.gps == 3, ] %*% beta.group3
y = x.beta + 5 * rnorm(n)
```

We're ready to split into train, validation and test sets. We will use people 1, 2 and 3 for training and validation (two-thirds train, one-third validation), and people 4 and 5 for testing.

```r
trn.index = groups < 4
val.sample = sample(1:sum(trn.index), 1/3 * sum(trn.index), replace = FALSE)

xtrain = x[trn.index, ][-val.sample, ]
ytrain = y[trn.index][-val.sample]
gpstrain = groups[trn.index][-val.sample]

xval   = x[trn.index, ][val.sample, ]
yval = y[trn.index][val.sample]
gpsval = groups[trn.index][val.sample]

xtest  = x[!trn.index, ]
ytest = y[!trn.index]
gpstest = groups[!trn.index]
```

First, we train a model to predict the person ID from the covariates. Then, we will use this model to predict similarity of the test set observations to the training set people.

```r
simmod = cv.glmnet(xtrain, as.factor(gpstrain), family = "multinomial")

class.preds = predict(simmod, xtest, type="response")[, , 1]
```

Because this example is simulated, we can measure the performance of our model on test data. In real settings, this would be impossible. Let's look at the confusion matrix comparing predicted group labels to true group labels.

```r
table(apply(class.preds, 1, which.max),
      hidden.gps[groups >= 4])
#>
#>       1   2   3
#>   1 258  34   3
#>   2  41  86  27
#>   3   0  35 116
```

Now, we can do pretraining as normal using the training set, with the person ID acting as the grouping.

```r
cvfit = cv.ptLasso(xtrain, ytrain, gpstrain,
                   type.measure = "mse",
                   group.intercepts = FALSE,
                   overall.lambda = "lambda.1se")
```

This is what performance would be if we only used the overall model and ignored that our data came from different people.

```r
overall.predictions = predict(cvfit$fitoverall, xtest)
assess.glmnet(overall.predictions, newy = ytest)$mse
#> lambda.1se
#>   30.57228
#> attr(,"measure")
#> [1] "Mean-Squared Error"
```

Now, let's see what performance is with pretraining. We'll use the value of α selected by cross-validation.

```
alphahat  = cvfit$alphahat
bestmodel = cvfit$fit[[which(cvfit$alphalist == alphahat)]]
cat("Chosen alpha is", alphahat, ".\n")
#> Chosen alpha is 0.9 .

offset = (1-alphahat) * predict(bestmodel$fitoverall, xtest, s = "lambda.1se")

# Get the prediction for all three classes for each test observation.
# This will be a matrix with three columns; one for each class.
pretrained.preds = do.call(cbind,
                           lapply(1:3,
                                  function(i) predict(bestmodel$fitpre[[i]],
                                                      xtest,
                                                      newoffset = offset)
                         )
)
```

Our pretrained predictions are the weighted combination of the predictions from `ptLasso` and the class predictions from `glmnet`:

```
assess.glmnet( rowSums(pretrained.preds * class.preds), newy = ytest)$mse
#> [1] 30.32199
#> attr(,"measure")
#> [1] "Mean-Squared Error"
```

We can use the same weighting approach to compute the MSE for the individual models:

```
individual.preds = do.call(cbind,
                              lapply(1:3,
                                     function(i) predict(bestmodel$fitind[[i]],
                                                         xtest,
                                                         type = "response")
                         )
)
assess.glmnet(rowSums(individual.preds * class.preds), newy = ytest)$mse
#> [1] 30.66575
#> attr(,"measure")
#> [1] "Mean-Squared Error"
```

Note that, while what we have done makes sense mathematically, we have found better empirical results if we instead train a supervised learning algorithm to make the final prediction $\hat{y}$ using the pretrained model predictions and the class similarity predictions as features. So, let's do that here, using our so-far-untouched validation set.

```
val.offset = predict(bestmodel$fitoverall, xval, s = "lambda.1se")
val.offset = (1 - alphahat) * val.offset
val.preds = do.call(cbind,
                    lapply(1:3, function(i) predict(bestmodel$fitpre[[i]],
                                                     xval,
                                                     newoffset = val.offset,
                                                     type = "response")
```

```r
                       )
)
val.class.preds = predict(simmod, xval)[, , 1]

pred.data = cbind(val.preds, val.class.preds, val.preds * val.class.preds)
final.model = cv.glmnet(pred.data, rowSums(val.preds * val.class.preds))

pred.data.test = cbind(pretrained.preds,
                        class.preds,
                        pretrained.preds * class.preds)
assess.glmnet(predict(final.model, pred.data.test), newy = ytest)$mse
#> lambda.1se
#>   30.44617
#> attr(,"measure")
#> [1] "Mean-Squared Error"
```

Let's look at performance of all models side-by-side:

```r
rd = function(x) round(x, 2)

cat("Overall model PSE: ",
    rd(assess.glmnet(overall.predictions, newy = ytest)$mse),
    "\n"
)
#> Overall model PSE:  30.57

cat("Individual model PSE: ",
    rd(assess.glmnet(rowSums(individual.preds*class.preds), newy = ytest)$mse),
    "\n"
)
#> Individual model PSE:  30.67

cat("Pretraining model PSE: ",
    rd(assess.glmnet(rowSums(pretrained.preds*class.preds), newy = ytest)$mse),
    "\n"
)
#> Pretraining model PSE:  30.32

cat("Pretraining model + final prediction model PSE: ",
    rd(assess.glmnet(predict(final.model,
                             cbind(pretrained.preds,
                                   class.preds,
                                   pretrained.preds * class.preds)
                       ),
              newy = ytest)$mse),
    "\n"
)
#> Pretraining model + final prediction model PSE:  30.45
```

In our simulated setting, including the final model did not help performance – however, we still recommend trying this with real data.