Using nonlinear bases

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

Suppose we have a dataset with features X and response y, where the relationship between X and y is a nonlinear function of the columns of X. Can we still use the lasso? Yes! We can *pretrain* our linear model using `xgboost` to obtain basis functions (features). Let's walk through an example.

0.1 Example 1: xgboost pretraining

We start by simulating data ($n = 1800$, $p = 1000$) with a continuous response. Our coefficients β are sparse; the first 200 entries will be drawn from a standard univariate normal, and the remainder are 0. We define y as $y = 1(X > 0)\beta + \epsilon$, where ϵ is noise; we hope that `xgboost` will learn the splits corresponding to $X > 0$.

```
set.seed(1234)

n = 1800; p = 1000; noise = 5;

x     = matrix(rnorm(n * p), nrow=n, ncol=p)
xtest = matrix(rnorm(n * p), nrow=n, ncol=p)

x.model     = 1*(x > 0)
xtest.model = 1*(xtest > 0)

beta = c(rnorm(200), rep(0, p-200))

y     = x.model %*% beta + noise * rnorm(n)
ytest = xtest.model %*% beta + noise * rnorm(n)

train.folds = sample(rep(1:10, n/10))
```

Now, we run `xgboost` to get our basis functions:

```
xgbfit      = xgboost(data=x, label=y, nrounds=200, max_depth=1, verbose=0)

x.boost     = predict(xgbfit, x, predleaf = TRUE) - 1
xtest.boost = predict(xgbfit, xtest, predleaf = TRUE) - 1
```

And we are ready for model fitting with `cv.glmnet`. Our two baselines are (1) a linear model that does not pretrain with `xgboost`, and (2) `xgboost`. We find that `glmnet` together with `xgboost` outperforms `glmnet` alone and `xgboost` alone.

Table 1: Coefficients for simulating data

	1-50	51-100	101-150	151-200	201-500
group 1	2	1	0	0	0
group 2	2	0	1	0	0
group 3	2	0	0	1	0

```
cvfit = cv.glmnet(x.boost, y, type.measure = "mse", foldid = train.folds)
cvfit.noboost = cv.glmnet(x, y, type.measure = "mse", foldid = train.folds)

cat("Prediction squared error - lasso with xgboost pretraining: ",
    assess.glmnet(cvfit, newx = xtest.boost, newy = ytest)$mse)
#> Prediction squared error - lasso with xgboost pretraining:  46.23225

cat("Prediction squared error - lasso without xgboost pretraining: ",
    assess.glmnet(cvfit.noboost, newx = xtest, newy = ytest)$mse)
#> Prediction squared error - lasso without xgboost pretraining:  60.68818

cat("Prediction squared error - xgboost alone: ",
    assess.glmnet(predict(xgbfit, xtest), newy = ytest)$mse)
#> Prediction squared error - xgboost alone:  49.47738
```

0.2 Example 2: xgboost pretraining with input groups

Now, let's repeat the above supposing our data have input groups. The only difference here is that we will use `cv.ptLasso` for our model instead of `cv.glmnet`, and we will use the group indicators as a feature when fitting `xgboost`.

We start by simulating data with 3 groups (600 observations in each group) and a continuous response. The coefficients for the groups are in Table 1.

As before, we will simulate y as $y = 1(X > 0)\beta + \epsilon$, only now we have a different β for each group.

```
set.seed(1234)

n = 1800; p = 500; k = 3;
noise = 5;

groups = groupstest = sort(rep(1:k, n/k))

x     = matrix(rnorm(n * p), nrow=n, ncol=p)
xtest = matrix(rnorm(n * p), nrow=n, ncol=p)

x.model     = 1*(x > 0)
xtest.model = 1*(xtest > 0)

common.beta = c(rep(2, 50), rep(0, p-50))
beta.1 = c(rep(0, 50),  rep(1, 50), rep(0, p-100))
beta.2 = c(rep(0, 100), rep(1, 50), rep(0, p-150))
beta.3 = c(rep(0, 150), rep(1, 50), rep(0, p-200))

y = x.model %*% common.beta + noise * rnorm(n)
y[groups == 1] = y[groups == 1] + x.model[groups == 1, ] %*% beta.1
```

```
y[groups == 2] = y[groups == 2] + x.model[groups == 2, ] %*% beta.2
y[groups == 3] = y[groups == 3] + x.model[groups == 3, ] %*% beta.3

ytest = xtest.model %*% common.beta + noise * rnorm(n)
ytest[groups == 1] = ytest[groups == 1] + xtest.model[groups == 1, ] %*% beta.1
ytest[groups == 2] = ytest[groups == 2] + xtest.model[groups == 2, ] %*% beta.2
ytest[groups == 3] = ytest[groups == 3] + xtest.model[groups == 3, ] %*% beta.3
```

Here are the dummy variables for our group indicators; we will use them to fit and predict with `xgboost`.

```
group.ids     = model.matrix(~as.factor(groups) - 1)
grouptest.ids = model.matrix(~as.factor(groupstest) - 1)
colnames(grouptest.ids) = colnames(group.ids)
```

Now, let's train `xgboost` and `predict` to get our new features. Note that we now use `max_depth = 2`: this is intended to allow interactions between the group indicators and the other features.

```
xgbfit      = xgboost(data=cbind(x, group.ids), label=y,
                      nrounds=200, max_depth=2, verbose=0)

x.boost     = predict(xgbfit, cbind(x, group.ids), predleaf = TRUE) - 1
xtest.boost = predict(xgbfit, cbind(xtest, grouptest.ids), predleaf = TRUE) - 1
```

Fit and predict two models trained with `cv.ptLasso`: one uses the xgboost features and the other does not.

```
cvfit = cv.ptLasso(x.boost, y, groups=groups, type.measure = "mse")
preds = predict(cvfit, xtest.boost, groups=groupstest, alphatype = "varying")
preds = preds$yhatpre

cvfit.noboost = cv.ptLasso(x, y, groups=groups, type.measure = "mse")
preds.noboost = predict(cvfit.noboost, xtest, groups=groupstest,
                        alphatype = "varying")
preds.noboost = preds.noboost$yhatpre
```

As before, we find that pretraining with xgboost improves performance relative to (1) model fitting in the original feature space and (2) xgboost alone.

```
cat("Prediction squared error - ptLasso with xgboost pretraining: ",
    assess.glmnet(preds, newy = ytest)$mse)
#> Prediction squared error - ptLasso with xgboost pretraining:  55.5542

cat("Prediction squared error - ptLasso without xgboost pretraining: ",
    assess.glmnet(preds.noboost, newy = ytest)$mse)
#> Prediction squared error - ptLasso without xgboost pretraining:  63.32061

cat("Prediction squared error - xgboost alone: ",
    assess.glmnet(predict(xgbfit, xtest), newy = ytest)$mse)
#> Prediction squared error - xgboost alone:  59.63781
```