eNetXplorer Vignette

Table of contents

1. About

The R package `eNetXplorer` is available under GPL-3 license at the CRAN repository. The package source is located at https://CRAN.R-project.org/package=eNetXplorer.

Authors: Julián Candia and John S. Tsang

Maintainer: Julián Candia <julian.candia@nih.gov>

2. Installation

To install to your default directory, type

```
install.packages("eNetXplorer")
```

For more installation options, type `help(install.packages)`.

3. eNetXplorer's Workflow

In order to describe `eNetXplorer`'s workflow, this Section presents the analysis pipeline applied to synthetic datasets; to further illustrate eNetXplorer's features, real datasets are distributed with the package as described in Sections below.

First, we create a function to generate data for a Gaussian (linear regression) model, which consists of:

- An input numerical matrix with `n_inst` instances or observations (as rows) and `n_pred` predictors or features (as columns).
- An input numerical vector of length `n_inst` with the observed response.

For normally distributed random variables, the following data-generating function will create a set of features correlated with the response according to any arbitrary, pre-defined **population** covariance matrix `covmat`:

```
data_gen_pop_covmat <- function(n_inst, covmat, seed=123) {
    library (expm);
    set.seed(seed)
    data <- matrix(rnorm(n_inst*ncol(covmat)),ncol=ncol(covmat))%*%sqrtm(covmat)
    predictor=data[,-1,drop=F]
    rownames(predictor) = paste0("Inst.",1:n_inst)
    colnames(predictor) = paste0("Feat.",1:(ncol(covmat)-1))
    list(response=data[,1],predictor=predictor)
}
```

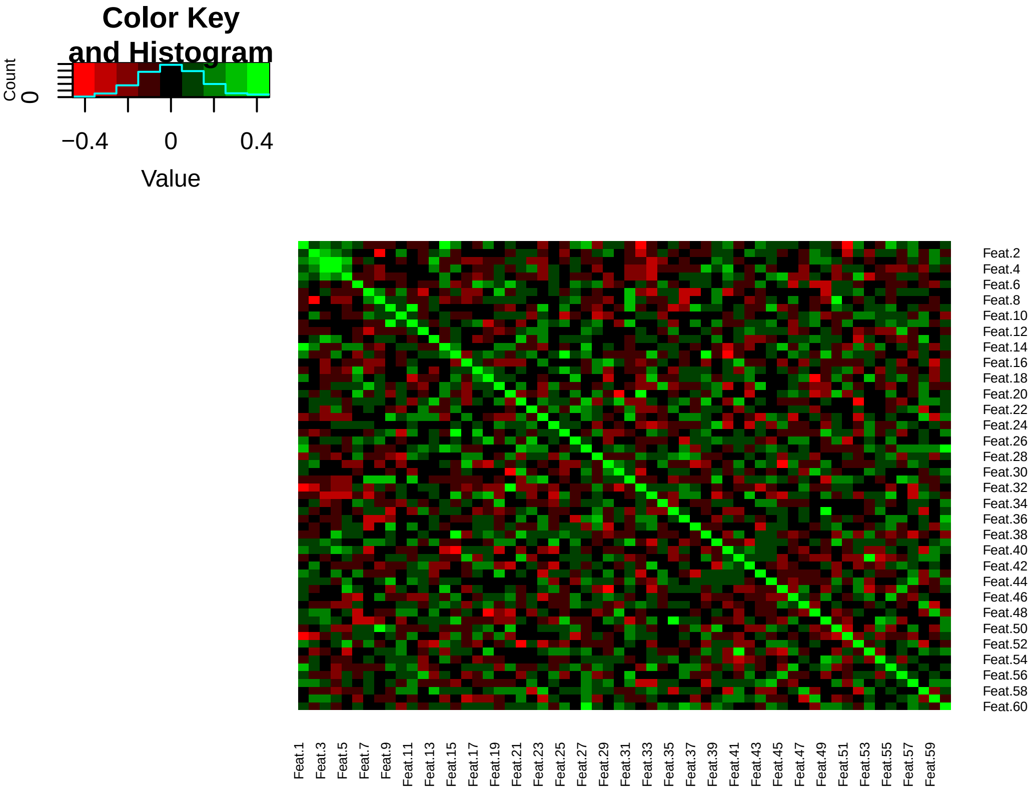
Here, we assume that the first row/column in `covmat` is the response, while the remaining variables correspond to the features. In order to generate covariant matrices with interesting properties, the following function will produce a block of features correlated with each other (where `r_block` is the intra-block Pearson's correlation) as well as correlated with the response (with correlation `r_resp`):

```
covmat_gen <- function(n_pred, block_size, r_resp, r_block) {
    covmat = matrix(rep(1.e-3,(n_pred+1)**2),ncol=(n_pred+1))
    for (i_pred in 1:block_size) {
        for (j_pred in (i_pred+1):(block_size+1)) {
            if (i_pred==1) {
                covmat[i_pred,j_pred] = r_resp
            } else {
                covmat[i_pred,j_pred] = r_block
            }
            covmat[j_pred,i_pred] = covmat[i_pred,j_pred]
        }
    }
    for (i_pred in 1:n_pred) {
        covmat[i_pred,i_pred] = 1
    }
    covmat
}
```

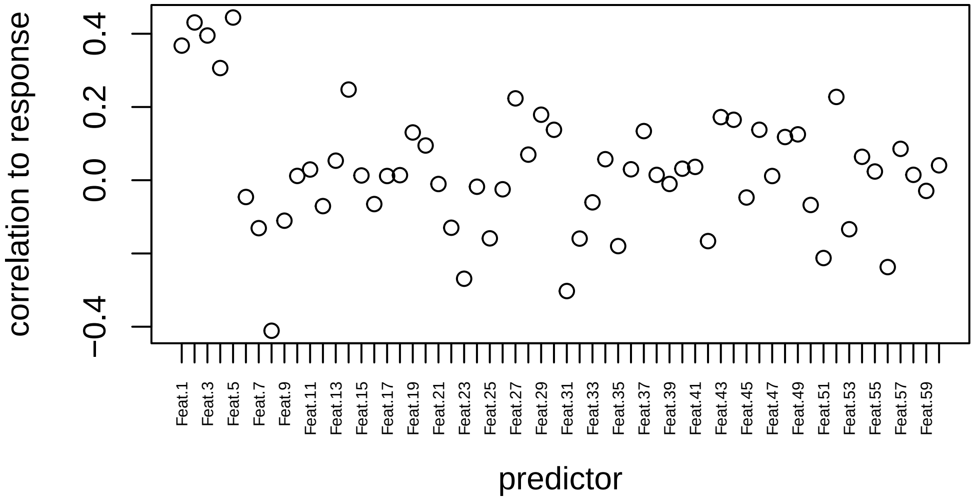
Let us use these two functions to generate predictor and response inputs:

```
data = data_gen_pop_covmat(n_inst=50, covmat_gen(n_pred=60, block_size=5, r_resp=0.5, r_block=0.35))
```

We examine the predictor correlation structure; as expected from our `covmat_gen` call, the block of features 1-5 is significantly intra-correlated, while the remaining features appear only weakly (and, in some cases, even negatively) correlated.



We also examine the correlation between predictors and the response; by design, the block of features 1-5 carries the largest positive correlation with the response.



Since the number of features is larger than the number of observations, we need to implement a regularized regression model. But which model? We know that ridge will fit a model where all predictors (including all the non-informative ones) will have non-zero contributions. Lasso, on the other end, will exclude most features and provide a minimal model representation. The elastic net allows us to scan the regularization path from ridge to lasso via the mixing parameter `alpha`. However, the following open questions remain:

- Which `alpha` represents the top-performing regularized model?
- What is the model-level statistical significance across `alpha`?
- What is the feature-level statistical significance of a given `alpha`-model?
- How does feature-level statistical significance change across `alpha`?

To address these questions, `eNetXplorer` generates an ensemble of null models (based on random permutations of the response) on a family of regularized models from ridge to lasso. First, we load the `eNetXplorer` package:

```
library(eNetXplorer)
```

Next, we run `eNetXplorer` on the datasets we just generated. The call to `eNetXplorer` with default parameters is:

```
fit_def = eNetXplorer(x = data$predictor, y = data$response, family = "gaussian")
```

Results can be made more precise by increasing the number of cross-validation runs (`n_run`) and the number of null-model response permutations per run (`n_perm_null`), as well as by choosing a smaller step in the path of `alpha` models:

```
fit = eNetXplorer(x = data$predictor, y = data$response, family = "gaussian", alpha = seq(0,
    1, by = 0.1), n_run = 1000, n_perm_null = 250, seed = 123)
```

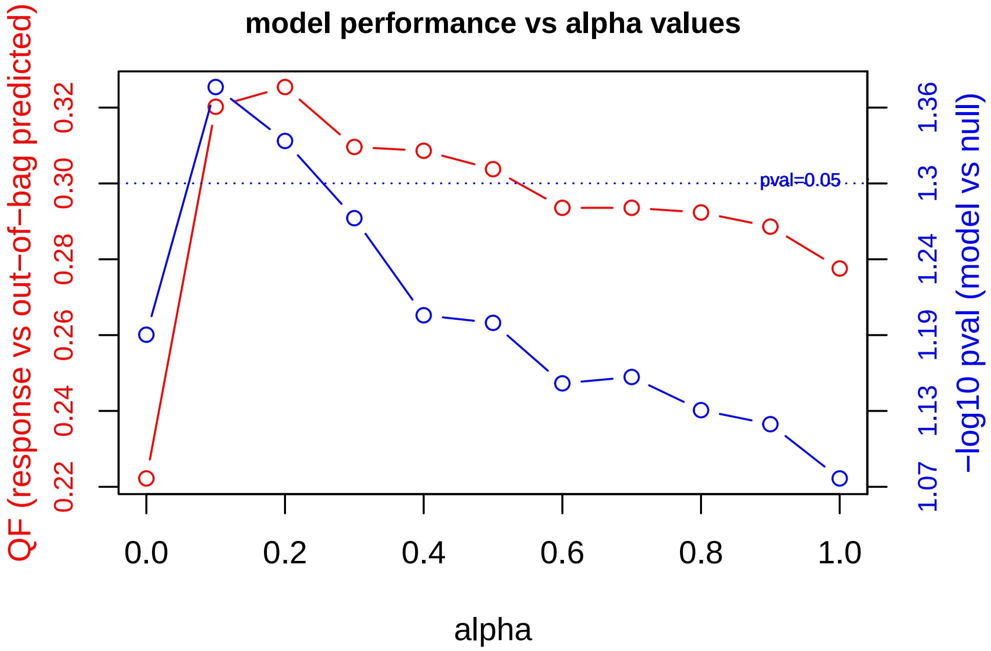
Function `summary` generates a brief report on the results; for each `alpha`, it displays the optimal `lambda` (obtained by maximizing a quality function over out-of-bag instances), the corresponding maximum value of the quality function, and the model significance (p-value based on comparison to permutation null models).

```
summary(fit)
```

```
## Call:
## eNetXplorer(x = data$predictor, y = data$response, family = "gaussian",
##     alpha = seq(0, 1, by = 0.1), seed = 123, n_run = 1000, n_perm_null = 250)
##
##     alpha lambda.max QF.est model.vs.null.pval
##  0.000000   3.724056 0.2222            0.06524 .
##  0.100000   0.697820 0.3202            0.04222 *
##  0.200000   0.382928 0.3254            0.04641 *
##  0.300000   0.293516 0.3096            0.05316 .
##  0.400000   0.200581 0.3086            0.06305 .
##  0.500000   0.168105 0.3038            0.06389 .
##  0.600000   0.140088 0.2936            0.07106 .
##  0.700000   0.125793 0.2936            0.07027 .
##  0.800000   0.115310 0.2923            0.07448 .
##  0.900000   0.097839 0.2886            0.07634 .
##  1.000000   0.092248 0.2776            0.08398 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A graphical display of model performance across `alpha` is provided by

```
plot(fit, plot.type="summary")
```

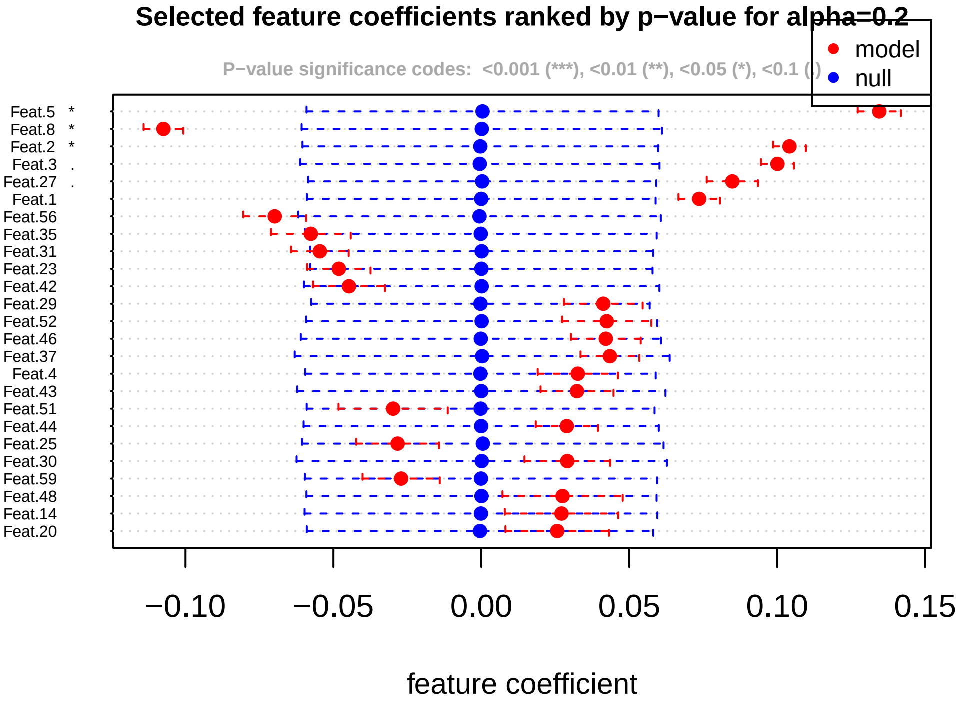


We observe that, at `p-value`<0.05, the most significant models are `alpha`=0.1 and 0.2; in terms of performance, the quality function (which, by default, is Pearson's correlation) evaluated between out-of-bag predictions and the response is maximized by the `alpha`=0.2 model. Let us examine this top-performing model in more detail.

Our next question is to determine the top features that play a role in the top-performing model. Following a similar strategy to that of `alpha`-model selection, statistical significance at the feature level is determined by comparison to permutation null models; see (Candia and Tsang 2019) for technical details.

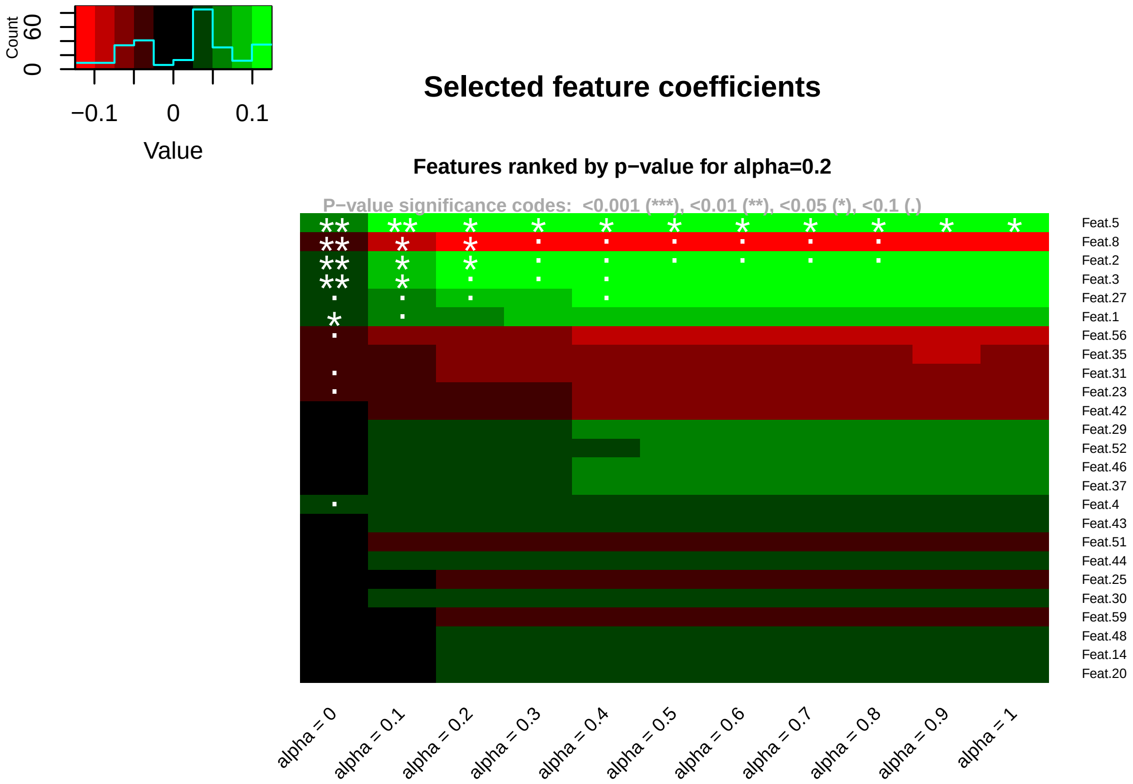
We generate a caterpillar plot of the top features based on their coefficients:

```
plot(fit, alpha.index = which.max(fit$model_QF_est), plot.type = "featureCaterpillar",
    stat = c("coef"))
```



We observe that features 5, 8, and 2 are selected by the regularized model with `alpha`=0.2 at `p-value`<0.05; features 3 and 27 are significant at `p-value`<0.1. Next, we aim to explore those same top features across the entire elastic net family: Which of them would still be selected under more stringent regularization criteria?

```
plot(fit, alpha.index = which.max(fit$model_QF_est), plot.type = "featureHeatmap",
    stat = c("coef"), notecex = 1.5)
```

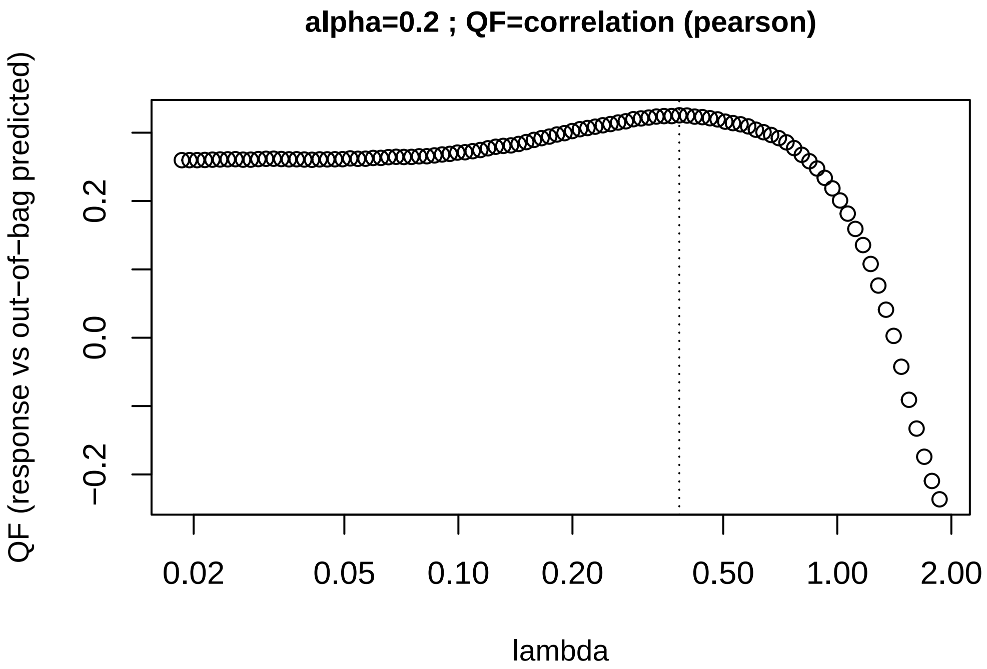


Here, we observe that more regularized models (`alpha`=0.5-0.8) favor features 5, 8 and 2 (at significance `p-value`<0.1); towards lasso (`alpha`=0.9,1), feature 5 is the only one selected at `p-value`<0.05-0.1. As expected, these models are more stringent on feature selection than less regularized (i.e. smaller `alpha`) models. It is also interesting to observe that lasso-like models remove feature redundancies: from the block of correlated features 1-5, only feature 5 is picked up as representative. This example illustrates some important characteristics of mixed-regularization model families:

- less regularized (smaller `alpha`) models promote redundancy; they benefit from borrowed information across significantly correlated predictors; they provide larger signatures, which are potentially more robust and resilient under measurement noise; they offer more opportunities for systems-level interpretation (e.g. downstream pathway analysis in the context of genomics).
- more regularized (larger `alpha`) models promote sparsity; they tend to pick just one predictor out of a set of correlated ones; they may facilitate interpretation with high-dimensional datasets and/or in the absence of systems-level annotations; they provide smaller signatures, which may be more useful in certain contexts (e.g. biomarker panels).

In order to gather more details regarding a particular solution, we plot the quality function across the range of values for the regularization parameter `lambda`:

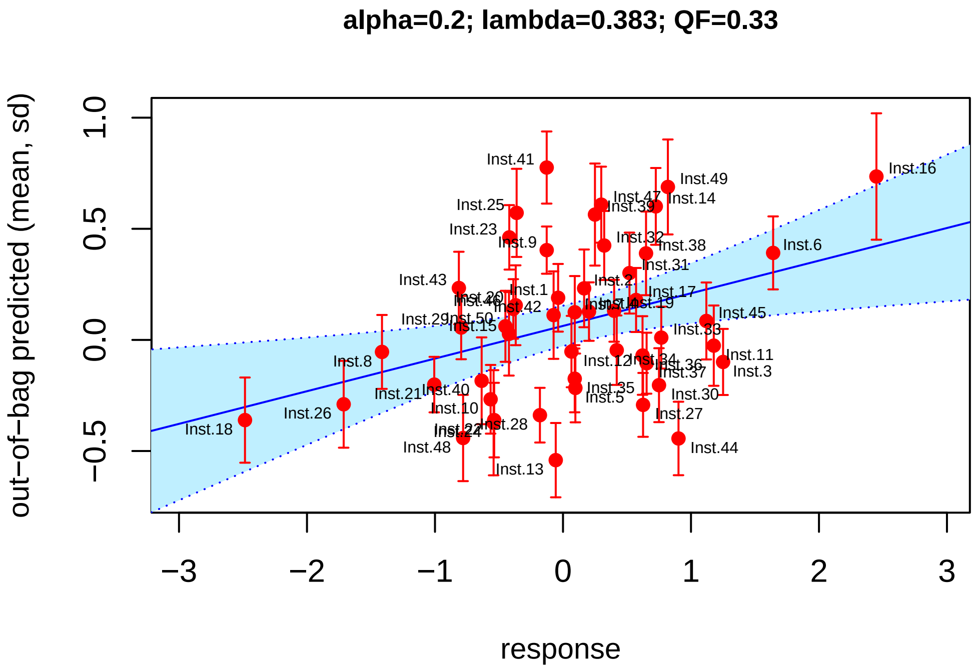
```
plot(fit, alpha.index = which.max(fit$model_QF_est), plot.type = "lambdaVsQF")
```



If so desired, eNetXplorer allows end-users to extend the number of lambda values (via nlambda) and/or extend their range while keeping the lambda density uniform in log scale (via nlambda.ext).

There may exist outlier instances that may require further examination; we generate a scatterplot of response vs out-of-bag predictions across all instances:

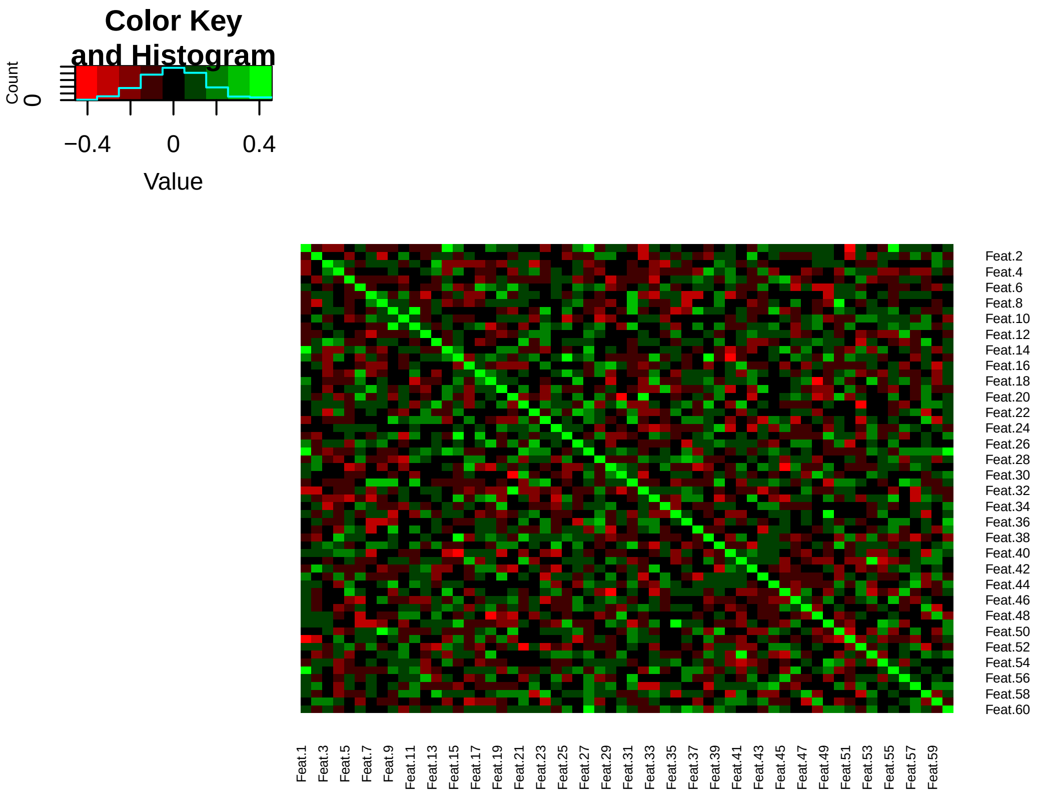
```
plot(fit, alpha.index = which.max(fit$model_QF_est), plot.type = "measuredVsOOB")
```



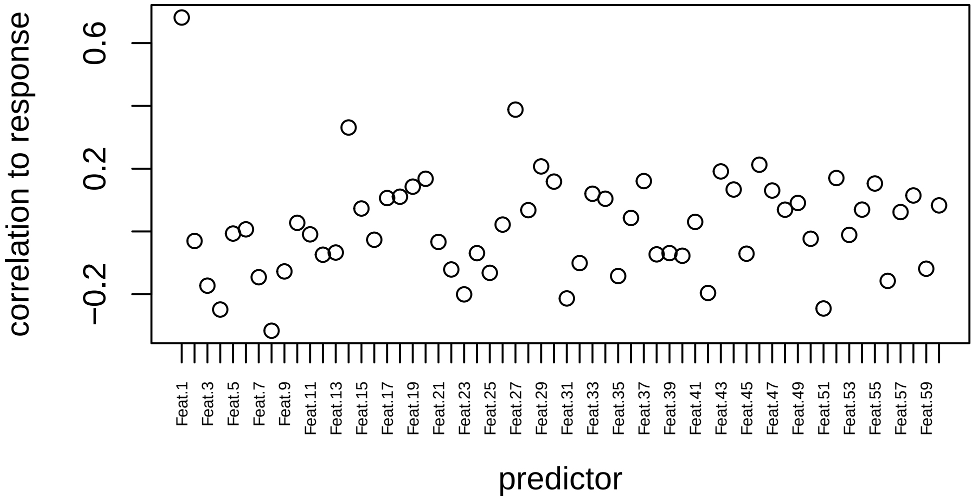
Naturally, model performance across `alpha` is strongly dependent on the structure of the input datasets. In the case example above, we purposefully generated a block of informative correlated predictors to highlight the characteristics of less regularized (smaller `alpha`) models and their ability to leverage borrowed information. Let us now generate input datasets with just one prevalent informative predictor:

```
data = data_gen_pop_covmat(n_inst=50, covmat_gen(n_pred=60, block_size=1, r_resp=0.7, r_block=0.35))
```

As before, we examine the predictor correlation structure:



Moreover, as before, we also examine the correlation between predictors and the response; by design, feature 1 is the most informative predictor:



We run `eNetXplorer` on the new datasets:

```
fit = eNetXplorer(x = data$predictor, y = data$response, family = "gaussian", alpha = seq(0,
    1, by = 0.1), n_run = 1000, n_perm_null = 250, seed = 123)
```

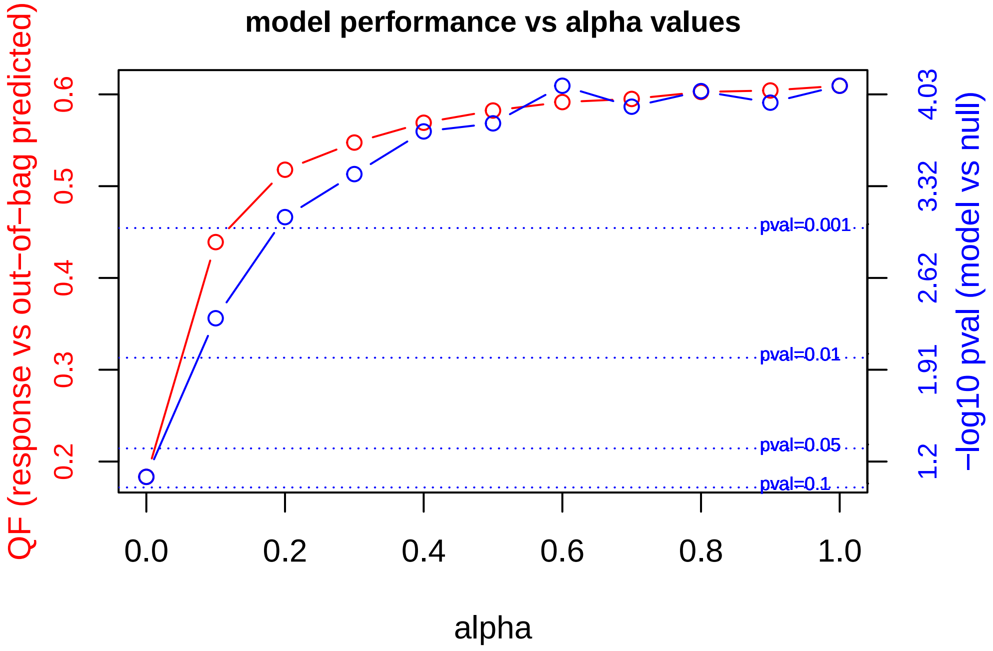
The summary table is:

```
summary(fit)
```

```
## Call:
## eNetXplorer(x = data$predictor, y = data$response, family = "gaussian",
##     alpha = seq(0, 1, by = 0.1), seed = 123, n_run = 1000, n_perm_null = 250)
##
##    alpha lambda.max QF.est model.vs.null.pval
##  0.00000    6.15154 0.1833           0.082884 .
##  0.10000    1.92280 0.4391           0.004960 **
##  0.20000    1.21315 0.5179           0.000824 ***
##  0.30000    0.88762 0.5475           0.000384 ***
##  0.40000    0.66571 0.5691           0.000180 ***
##  0.50000    0.53257 0.5824           0.000156 ***
##  0.60000    0.42364 0.5916            8.0e-05 ***
##  0.70000    0.38041 0.5950           0.000116 ***
##  0.80000    0.33286 0.6026            8.8e-05 ***
##  0.90000    0.29587 0.6042           0.000108 ***
##  1.00000    0.27896 0.6094            8.0e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We generate the plot of model performance across `alpha`:

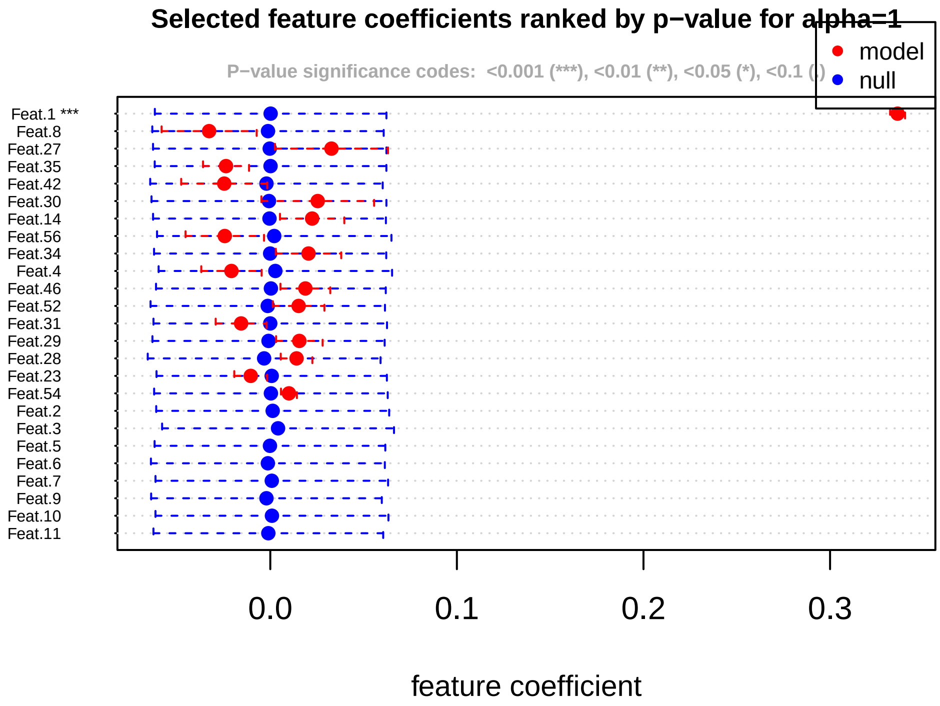
```
plot(fit, plot.type="summary")
```



Here, we observe that the quality function appears to increase monotonically with `alpha`; the maximum corresponds to the lasso solution, `alpha`=1.

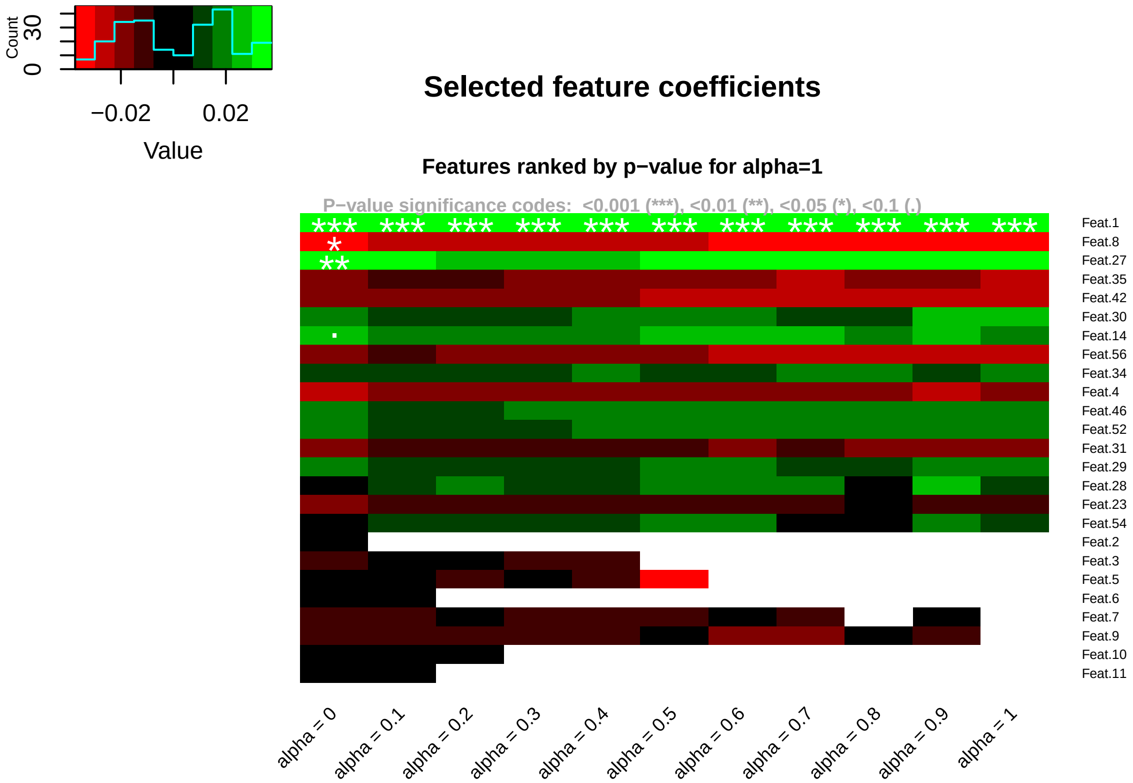
We generate a caterpillar plot of the top features based on their coefficients:

```
plot(fit, alpha.index = which.max(fit$model_QF_est), plot.type = "featureCaterpillar",
    stat = c("coef"))
```



We observe that only feature 1 is selected by lasso. Are there other features that would be selected under less stringent regularization criteria?

```
plot(fit, alpha.index = which.max(fit$model_QF_est), plot.type = "featureHeatmap",
    stat = c("coef"), notecex = 1.5)
```



We find that, except for features 8, 27 and 14 for ridge regularization (`alpha`=0), feature 1 appears dominant all across the regularization range. This, indeed, reflects the covariance structure we chose to generate the data.

4. Datasets

The `eNetXplorer` package provides three datasets of biological interest:

- `H1N1_Flow`, comprised of longitudinal cell population frequencies and titer responses upon H1N1 vaccination;
- `Leukemia_miR`, which contains microRNA (miR) expression data from cell lines and primary (patient) samples classified by different acute leukemia phenotypes, as well as normal control samples sorted by cell type; and
- `breastCancerSurv`, which has survival data and microarray data from a panel of seven genes reported to be associated with breast cancer clinical outcome.

4.1 `H1N1_Flow`

The `H1N1_Flow` dataset comprises data from a cohort of healthy subjects vaccinated against the influenza virus H1N1 (Tsang et al. 2014). Using five different 15-color flow cytometry stains for T-cell, B-cell, dendritic cell, and monocyte deep-phenotyping, 113 cell population frequencies were measured longitudinally pre- (days -7, 0) and post-vaccination (days 1, 7, 70) on a cohort of 49 healthy human subjects (F=31, M=18, median age=24). Cell populations were manually gated and expressed as percent of parent. Samples and cell populations were filtered independently for each timepoint; samples were excluded if the median of the fraction of viable cells across all five tubes was <0.7, while cell populations were excluded if >80% of samples had <20 cells. Data were log10-transformed and pooled across all timepoints, then adjusted for age, gender, and ethnicity effects. For each timepoint, a numerical matrix of predictors is provided with subjects as rows and cell populations as columns. The response is the adjusted maximum fold change (adjMFC) of serum titers at day 70 relative to baseline, as defined in (Tsang et al. 2014). Two versions of the serum titer response are provided in the package; one as a numerical vector and the other one as a categorical vector discretized into low ("0"), intermediate ("1") and high ("2") response classes. A metadata file with cell population annotations is also provided.

To load the dataset:

```
data(H1N1_Flow)
```

4.2 `Leukemia_miR`

The `Leukemia_miR` dataset comprises data of human microRNA (miR) expression of 847 miRs from 80 acute myeloid (AML) and acute lymphoblastic (ALL) leukemia cell lines, 60 primary (patient) samples, and 50 normal control samples sorted by cell type (CD34+ HSPC, Granulocytes, Monocytes, T-cells and B-cells) (Tan et al. 2014; Candia et al. 2015). Acute lymphoblastic leukemia samples are further classified by B-cell (B-ALL) and T-cell (T-ALL) subphenotypes. Two dataset versions are provided: the full dataset `Leuk_miR_full` (190 samples x 847 miRs) and the filtered dataset `Leuk_miR_filt` (140 samples x 370 miRs). A numerical matrix of predictors is provided with samples as rows and miRs as columns. Two categorical response vectors are provided for binomial (AML, ALL) and multinomial (AML, B-ALL, T-ALL) classification.

To load the dataset:

```
data(Leukemia_miR)
```

To filter the full dataset (and recapitulate the filtered data provided by `Leuk_miR_filt`):

```
expr_full = Leuk_miR_full$expression_matrix
miR_filter = rep(F,nrow(Leuk_miR_full$miR_metadata))
miR_filter[apply(expr_full,2,mean)>1.2] = T
sample_filter = rep(T,nrow(Leuk_miR_full$sample_metadata))
sample_filter[Leuk_miR_full$sample_metadata$sample_class=="Normal"] = F
expr_filtered = expr_full[sample_filter,miR_filter]
miR_filtered = Leuk_miR_full$miR_metadata[miR_filter,]
sample_filtered = Leuk_miR_full$sample_metadata[sample_filter,]
```

4.3 `breastCancerSurv`

The `breastCancerSurv` dataset consists of gene expression microarray data for a panel of 7 genes and 319 breast cancer patients, for which also survival information is available. A numerical matrix of predictors is provided with samples as rows and genes as columns. The response is a numerical matrix with columns named "time" and "status," where the latter is a binary indicator of death (1) or right-censoring (0). It should be noticed that this exact formatting of the response matrix is required; the function `Surv()` in package `survival` produces such a matrix.

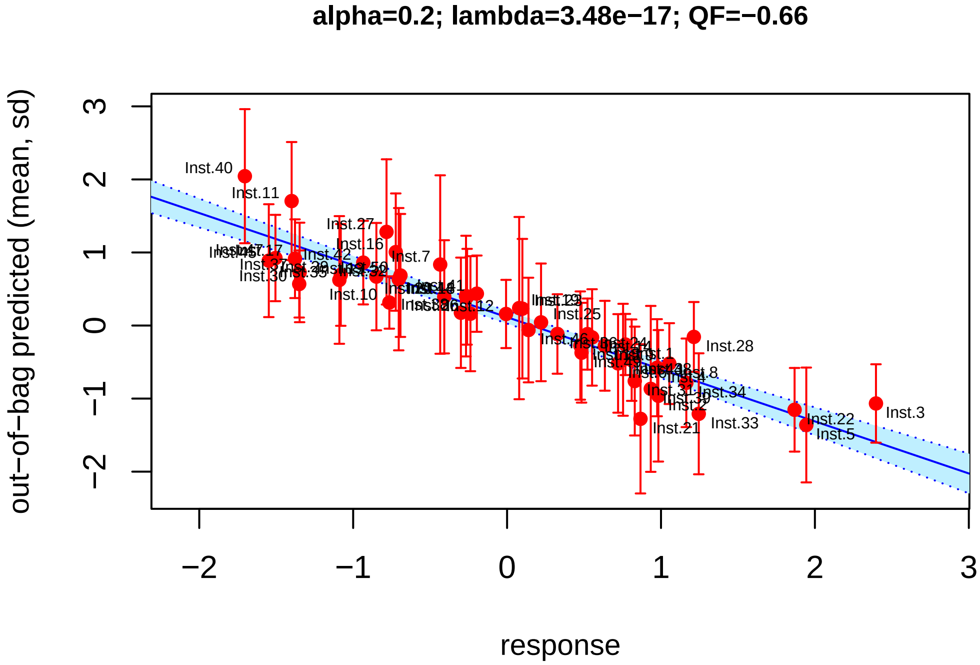
To load the dataset:

```
data(breastCancerSurv)
```

5. Why is my linear regression slope negative?

You may notice that, when running a linear regression model, the correlation between out-of-bag (OOB) predictions and the (true) response values may display a negative slope. This is typically observed when the set of model predictors is not (or is only weakly) informative of the response. For example, running `eNetXplorer` after generating a dataset of 50 instances with 20 predictors whose correlation with the response is exactly zero, the correlation between OOB predictions and the response is:

```
plot(eNet, alpha.index = which.max(eNet$model_QF_est), plot.type = "measuredVsOOB")
```



which has a negative slope! Shouldn't the slope be zero? Well, as it turns out, we should expect the slope to be negative because of the phenomenon known as "regression to the mean." The in-bag fit will generally result in a spurious positive slope due to finite-size sampling bias; relative to the OOB data, this spurious trend will thus be negatively correlated. As a very straightforward illustration of this phenomenon, let us consider a step-by-step, simple example of univariate regression.

First, the following data-generating function will create a set of features correlated with the response according to any arbitrary, pre-defined **sampling** covariance matrix `covmat`:

```
data_gen_sampl_covmat <- function(n_inst, covmat) {
    library (expm)
    mat <- matrix(rnorm(n_inst*ncol(covmat)),ncol=ncol(covmat))
    data = mat%*%sqrtm(solve(cov(mat)))%*%sqrtm(covmat)
    predictor=data[,-1,drop=F]
    rownames(predictor) = paste0("Inst.",1:n_inst)
    colnames(predictor) = paste0("Feat.",1:(ncol(covmat)-1))
    list(response=data[,1],predictor=predictor)
}
```

In passing, let us mention that this function requires the number of instances to be larger than the number of variables, whereas `data_gen_pop_covmat` introduced earlier, which generates data based on a pre-defined **population** covariance matrix, does not have this limitation.

Next, we generate the fold template for cross-validated linear regression:

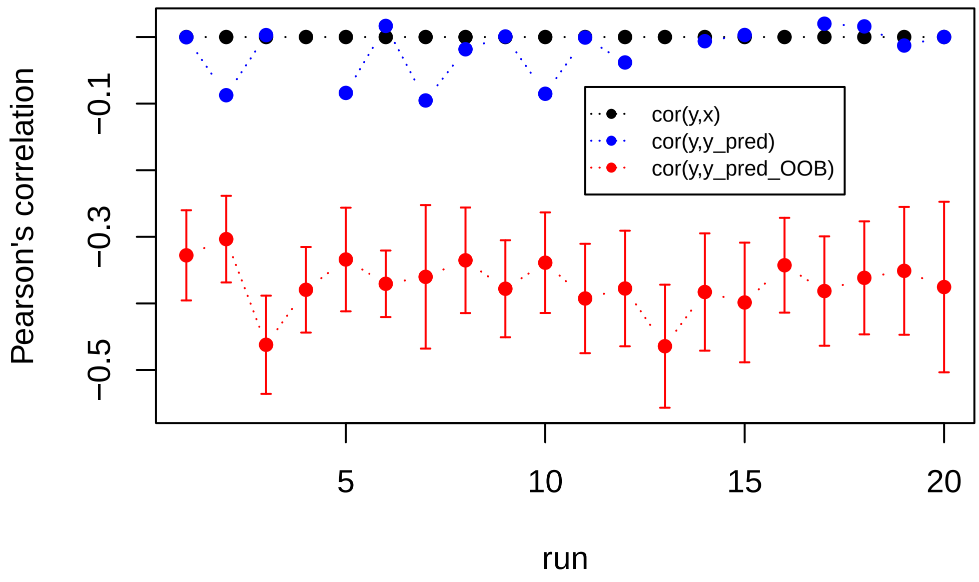
```
n_inst=50
n_fold = 5
foldid = NULL
fold_size = floor(n_inst/n_fold)
for (i_fold in 1:n_fold) {
    foldid = c(foldid,rep(i_fold,fold_size))
}
fold_rest = n_inst%%n_fold
if (fold_rest>0) {
    for (i_fold in 1:fold_rest) {
        foldid = c(foldid,i_fold)
    }
}
```

In our next step, we randomly generate `n_run` univariate synthetic datasets; for each of them, the response is designed to have zero correlation with the predictor variable. For each dataset, we perform linear regression for the full dataset, as well as `n_rdm` cross-validated linear regressions.

```
n_run = 20
n_rdm = 10
set.seed(123)
cor_data = rep(NA,n_run)
cor_pred_full = rep(NA,n_run)
cor_pred_OOB_mean = rep(NA,n_run)
cor_pred_OOB_sd = rep(NA,n_run)
covmat=matrix(c(1,0,0,1),ncol=2)
for (i_run in 1:n_run) {
    data_gen = data_gen_sampl_covmat(n_inst,covmat)
    data = data.frame(x=data_gen$predictor[,1],y=data_gen$response)
    cor_data[i_run] = cor(data$x,data$y)
    fit = lm("y~x",data)
    y_pred = predict(fit,data)
    cor_pred_full[i_run] = cor(data$y,y_pred)
    # we generate random folds
    cor_pred_OOB = rep(NA,n_rdm)
    for (i_rdm in 1:n_rdm) {
        foldid_rdm = sample(foldid)
        y_pred_rdm = rep(NA,n_inst)
        for (i_fold in 1:n_fold) {
            IB = foldid_rdm!=i_fold
            fit = lm("y~x",data[IB,])
            y_pred_rdm[!IB] = predict(fit,data[!IB,])
        }
        cor_pred_OOB[i_rdm] = cor(data$y,y_pred_rdm)
    }
    cor_pred_OOB_mean[i_run] = mean(cor_pred_OOB)
    cor_pred_OOB_sd[i_run] = sd(cor_pred_OOB)
}
```

Finally, we plot the results.

```
color = c("black", "blue", "red")
x = 1:n_run
range_y = range(cor_data, cor_pred_full, cor_pred_OOB_mean + cor_pred_OOB_sd, cor_pred_OOB_mean -
    cor_pred_OOB_sd, na.rm = T)
plot(range(x), range_y, type = "n", xlab = "run", ylab = "Pearson's correlation",
    cex.lab = 1)
lines(x, cor_data, type = "b", pch = 16, col = color[1], lty = 3)
lines(x, cor_pred_full, type = "b", pch = 16, col = color[2], lty = 3)
lines(x, cor_pred_OOB_mean, type = "b", pch = 16, col = color[3], lty = 3)
arrows(x, cor_pred_OOB_mean - cor_pred_OOB_sd, x, cor_pred_OOB_mean + cor_pred_OOB_sd,
    col = color[3], length = 0.025, angle = 90, code = 3)
legend_txt = c("cor(y,x)", "cor(y,y_pred)", "cor(y,y_pred_OOB)")
legend(11, -0.075, legend_txt, pch = 16, lwd = 1, col = color, lty = 3, cex = 0.7,
    pt.cex = 0.7)
```



The x and y variables were designed to have zero correlation; the black plot just serves as a sanity check. As expected, we can't predict y from x; therefore, the correlation between the predicted y and the true y is essentially zero (blue plot). However, when using a cross-validation framework, the correlation between the OOB-predicted y and the true y is distinctly negative (red plot).

6. Notes on the algorithm's efficiency

eNetXplorer has been successfully tested and used on datasets with hundreds and up to 1,000+ features. However, the algorithm's efficiency degrades noticeably as the number of features is increased. In order to analyze high-dimensional datasets with thousands or, even, tens of thousands of features (e.g. RNA-Seq), we advise implementing dimensional-reduction preprocessing approaches. Useful strategies, for instance, are: (i) to compute the average (or, alternatively, the first principal component) of gene modules, which may be based on biological annotations (e.g. pathways, Gene Ontology classifications, etc.), previous meta-studies (Chaussabel et al. 2008; Li et al. 2014; Weiner 3rd and Domaszewska 2016) or calculated *de novo* (Langfelder and Horvath 2008); or (ii) to filter out low-expression genes, then select the most variable genes across the cohort (ranked by variance or mean absolute deviation).

It should also be noticed that the efficiency of linear regression (Gaussian) and binomial classification models is far superior than that of multinomial classification. Therefore, whenever possible, we advise merging class labels (to turn a multinomial problem into a binomial one) or, in scenarios with multiple ordinal classes, converting the multinomial problem into a linear regression one.

Running time is linear on n_run (number of runs) and n_perm_null (number of random null-model permutations of the response per run). Default values were chosen with the purpose to achieve a balanced tradeoff between speed and statistical accuracy. We suggest running exploratory eNetXplorer models with default values (or adjusted as needed to generate relatively quick results), then re-run with more statistics to obtain more accurate estimates. Depending on the size and complexity of the dataset, as well as on the type of generalized linear model and parameters chosen, it is not uncommon to run eNetXplorer jobs that could take several hours, or even several days, to complete. Thus, we advise to start small (sacrificing statistical accuracy to shorten running time) and adjust parameters incrementally.

In addition, it is possible to run alpha values in smaller groups, or individually, and merge the resulting eNetXplorer objects. See Section on Parallelization below.

7. Parallelization

Upon sequential or parallel execution of two or more eNetXplorer runs with different values of the mixing parameter alpha, and assuming the objects from those runs have been saved, the mergeObj() function creates a new eNetXplorer object that merges the alpha values. It currently supports linear (gaussian), logistic (binomial), and Cox regression models.

It is possible to significantly optimize running time by executing individual alpha runs in parallel, saving individual eNetXplorer objects to be later merged at the end of the process. Be warned that, if executed, the code below will create objects in the current directory.

```
library(furrr)
data(QuickStartEx)
alpha_values = seq(0,1,by=0.2)
future::plan(multiprocess) # Set up parallel processing
eNet = alpha_values %>% furrr::future_map(~ eNetXplorer(
  x=QuickStartEx$predictor, y=QuickStartEx$response,family="gaussian",
  n_run=20,n_perm_null=10,save_obj=T,dest_obj=paste0("eNet_a",.x,".Robj"),alpha=.x))
mergeObj(paste0("eNet_a",alpha_values,".Robj"))
```

Special thanks to Adam Bartonicek (University of Otago, New Zealand) for contributing this solution.

8. Errata

In Figure 5 of our `eNetXplorer` paper (Candia and Tsang 2019):

- The caption for panels (**b**,**d**) should read "Boxplots showing the fraction of runs in which the prediction agrees with the outcome for a given patient."
- The y-axis for panels (**b**,**d**) should read "frequency of OOB correct predictions." The y-axis label in function `plotMeasuredVsOOB()` for categorical models has been fixed accordingly in version 1.1.0.

Special thanks to John Wiedenhöft (Universität Göttingen, Germany) for pointing out these errors.

9. Summary

- `eNetXplorer` addresses key questions regarding model regularization and feature selection based on permutation-based statistical significance tests. Results are provided in the form of standard plots, summary statistics and output tables.
- As illustrated by synthetic datasets, which were generated from covariance matrices involving normally distributed features and response distributions, `eNetXplorer` workflows are generally applicable to model any datasets consisting of `n_inst` observations across `n_pred` predictors or features that aim to explain a set of `n_inst` response values.
- Regularization models are particularly useful in scenarios involving a large number of features (even larger than the number of observations) and/or sets of significantly correlated features. These scenarios are typical of datasets generated by current technologies in molecular and cellular biology, but applications to other data rich environments are certainly possible.

References

Candia, J, S Cherukuri, Y Guo, K A Doshi, J R Banavar, C I Civin, and W Losert. 2015. "Uncovering Low-Dimensional, miR-Based Signatures of Acute Myeloid and Lymphoblastic Leukemias with a Machine-Learning-Driven Network Approach." *Converg Sci Phys Oncol* 1: 025002.

Candia, J, and J S Tsang. 2019. "eNetXplorer: An r Package for the Quantitative Exploration of Elastic Net Families for Generalized Linear Models." *BMC Bioinformatics* 20: 189.

Chaussabel, D, C Quinn, J Shen, P Patel, C Glaser, N Baldwin, D Stichweh, et al. 2008. "A Modular Analysis Framework for Blood Genomics Studies: Application to Systemic Lupus Erythematosus." *Immunity* 29: 150.

Langfelder, P, and S Horvath. 2008. "WGCNA: An r Package for Weighted Correlation Network Analysis." *BMC Bioinformatics* 9: 559.

Li, S, N Rouphael, S Duraisingham, S Romero-Steiner, S Presnell, C Davis, DS Schmidt, et al. 2014. "Molecular Signatures of Antibody Responses Derived from a Systems Biology Study of Five Human Vaccines." *Nat Immunol* 15: 195.

Tan, Y S, M Kim, T J Kingsbury, C I Civin, and W C Cheng. 2014. "Regulation of Rab5c Is Important for the Growth Inhibitory Effects of MiR-509 in Human Precursor-b Acute Lymphoblastic Leukemia." *PLoS One* 9: e111777.

Tsang, J S, P L Schwartzberg, Y Kotliarov, A Biancotto, Z Xie, R N Germain, E Wang, et al. 2014. "Global Analyses of Human Immune Variation Reveal Baseline Predictors of Postvaccination Responses." *Cell* 157: 499–513.

Weiner 3rd, J, and T Domaszewska. 2016. "Tmod: An r Package for General and Multivariate Enrichment Analysis." *PeerJ Preprints* 4: e2420v1.