Essential Regression Vignette - parseRun and plainER

Housekeeping

Before beginning, please ensure that the following packages are installed and libraries are loaded. The Essential Regression package is available for downloading using the following code:

```
devtools::install_github("Hanxi-002/EssReg", auth_token = "TOKEN_HERE")
```

The other packages can be installed from CRAN.

```
library(matrixcalc)
library(readr)
library(ROCR)
library(e1071)
library(dplyr)
library(doParallel)
library(foreach)
library(scales)
library(doRNG)
library(matlib)
library(BAS)
library(gmp)
library(EssReg)
```

We also recommend running some of these function in parallel. We prepare the environment for this using the following code:

```
cores <- detectCores() ## detect available number of cores, can be set manually as well
registerDoParallel(cores)
```

Data Set-Up

Essential Regression accepts a data matrix ($\mathbf{X}$) of dimensions $n \times p$, where n is the number of samples/patients/instances and p is the number of features/variables, and a response vector ($\mathbf{Y}$) of dimension n that can be either categorical or continuous.

This implementation of Essential Regression and its related functions all assume properly formatted and processed data. This data should be raw (non-standardized, non-normalized) as Essential Regression performs the standardization steps itself. However, this data should be pre-processed and cleaned. Missing values should be dealt with in an appropriate fashion (i.e. imputation, removal, etc.). Additionally, the structure and sparsity of $\mathbf{X}$ should be explored to ensure that the column-wise standard deviations do not equal zero and that there are no majority zero columns or rows.

As an example, we use a small dataset. $\mathbf{X}$, available for download here, has 14 samples and 200 features, and $\mathbf{Y}$, available for download here, has the responses from the 14 samples.

We provide a function, `parseRun()`, that accepts a .yaml file so that you do not need to ever actually call the `plainER()` function itself. However, we provide a walkthrough of both functions for reference.

Using `parseRun()`

`parseRun()` accepts a single argument, `yaml_path`, which is a string path to the `.yaml` file used for configuration of the run. We provide an example file that is available for download here.

This file is of the format:

```
---
x_path: x.csv             # path to .csv file for data matrix
y_path: y.csv             # path to .csv file for response vector
out_path: parseRun_out/   # path to directory for saving results
y_factor: FALSE           # is y categorical?
y_levels: NULL             # if y is ordinal, provide the levels
lambda: 0.1               # lambda
delta: [0.01, 0.05, 0.1] # delta
rep_cv: 10                # number of replicates for delta cross-validation
alpha_level: 0.05         # alpha level for confidence intervals
thresh_fdr: 0.2           # false discovery rate thresholding p-value cutoff
```

Notice that the `.yaml` file shares many of the same arguments that are used in `plainER()` (`out_path`, `lambda`, `delta`, `rep_cv`, `alpha_level`, and `thresh_fdr`). These should all be specified in the same way as they are in a call to `plainER()` with two small changes. Paths should not include quotations marks, and a vector of ranges for `delta` should use square brackets rather than parentheses (`[0.01, 0.02, 0.03]`). You will have to write out the sequence by hand, unfortunately, because `seq()` does not work. We do not specify `sigma` in the `.yaml` file; `sigma` is automatically set to `NULL` and the sample correlation matrix is calculated for you.

- `x_path`: A string path to the `x` data.
- `y_path`: A string path to the `y` data.
- `out_path`: A string path to an output directory that will be created within the pipeline. This MUST end in a forward slash (`/`).
- `y_factor`: A boolean flag indicating whether the response is categorical (AKA factor data).
- `y_levels`: A vector indicating the order of the levels of the response if `y` is ordinal. If there is no order, then set `y_levels` to `NULL`, and if there is an order, then provide a vector in the format `[level1, level2, level3]`. Make sure that you list all possible levels of `y`!
- `lambda`: A numerical value used in Essential Regression that controls the sparsity of the latent factors.
- `delta`: A numerical value or a vector used in Essential Regression that controls the number of latent factors found.
- `rep_cv`: An integer indicating the number of replicates to perform when cross-validating to find `delta`. This is only used when `delta` is specified with a vector. Set this to any number if only using a single value for `delta`.
- `alpha_level`: The value used for confidence intervals when estimating the coefficients for the latent factor regression within Essential Regression.
- `thresh_fdr`: The false discovery rate threshold on $\hat{\Sigma}$. We use this value as the upper limit for the p-values of the entries of the sample correlation matrix - if a given p-value is less than `thresh_fdr`, the entry is set to 0 in in $\hat{\Sigma}$.

Function Output

`parseRun()` will save its results as an `.rds` file in `out_path` named `er_input_DELTA.rds` where `DELTA` is the value for `delta` used. The function also saves heatmaps of $\hat{\Sigma}$ and the thresholded version of $\hat{\Sigma}$.

- `K`: The number of clusters identified by LOVE.

- `A`: $\hat{A}$, the estimated allocation matrix that indicates the latent cluster membership for all features $[p \times K]$.
- `C`: $\hat{C}$, the estimated covariance matrix of $\mathbf{Z}$ $[K \times K]$.
- `I_clust`: $\hat{I}$ as a list where each entry, i, is a vector containing the features that are found in latent cluster i.
- `I`: $\hat{I}$ as a vector of all features that are included in the latent clusters.
- `Gamma`: Γ, the estimated covariance of the error terms, W, from the factorization of $\mathbf{X}$ into $A\mathbf{Z} + W$.
- `beta`: β, the estimated coefficients for the regression on the latent variables, $\mathbf{Y} = \beta\mathbf{Z} + \epsilon$.
- `beta_conf_int`: The confidence intervals for the estimation of β according to the signficance level, α, specified by `alpha_level`.
- `beta_var`: The variance of the estimates of β.
- `pred`: A list containing the Essential Regression predictor ($\hat{\theta}_{ER}$), $\hat{\Theta}$, as well as the predicted values ($\mathbf{Y}$) made using the predictor.
- `opt_lambda`: The value of λ used.
- `opt_delta`: The value of δ used.
- `Q`: $\hat{\theta}_{ER}^{\top} \times \mathbf{X}^{\top}$ $[p \times K]$
- `thresh_sigma`: The matrix, $\hat{\Sigma}$, after thresholding according to `thresh_fdr`.

Running `parseER()`

Now that we understand the function specification, we can run Essential Regression. If you are copying this code, you will need to change `yaml_path` to the path to the `.yaml` file's location on your machine.

```
parseRun(yaml_path = "path/to/yaml") ## change path
```

Inspection of Results

Our results from running the above code can be downloaded here. Once you have downloaded our results and run the above code yourself, load in our results and compare it with your run. They should be EXACTLY the same.

```
comp_run <- readRDS("path/to/downloaded/run") ## change path
```

If working in RStudio, you can simply click on the objects (`comp_run`, or whatever you named the objects when running/loading) in the `Environment` window on the right side of the screen. If you prefer to work with code, you can use the variable access operator in R (`$`) to look at the results. `parseRun()` saves a list with lots of items, so use `names()` to find their names to use with `$`.

For example, if you want to look at the number of clusters in `comp_run`, you can do `names(comp_run)` to check what is inside `comp_run`. Remember from the above section that `K` is the number of clusters, so we would then just type `comp_run$K`, and R will print the number of clusters in the console.

Be aware that some results, such as the `A` matrix, can be quite large, and you will not want to print these to the console. It is better to use `View()` for these larger matrices/data frames so you can see them in a separate window — just do `View(comp_run$A)`.

You can also use the `readER()` function to look at the clusters in the output of Essential Regression. To look at the clusters for `comp_run`, you would do `clusts <- readER(x, comp_run)`. You can then look at the pure and mixed variables as well as the cluster membership information by using `$` on `clusts`. We provide an example below.

Chosen Delta

We can peek at what δ was used (even though we know from the output name).

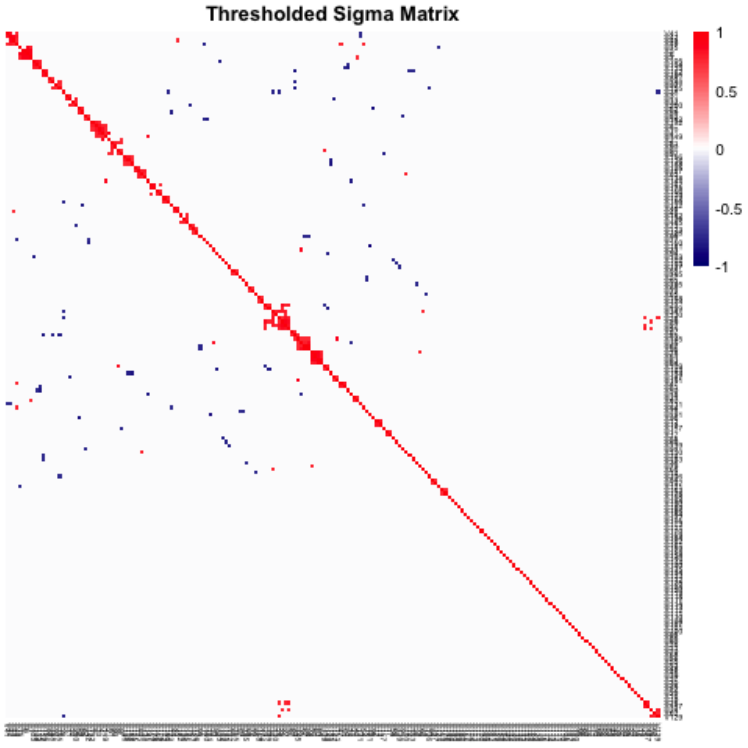
```
comp_run$opt_delta
```

```
## [1] 0.05
```

Thresholded $\hat{\Sigma}$

We use the `makeHeatmap()` function to plot the correlation matrix. `mat` is the matrix itself, which is contained in the Essential Regression output list. `title` is a title for the heatmap. `cluster` is a boolean flag indicating whether to perform hierarchical clustering on the rows and columns of the hatmap. `names` is a boolean flag indicating whether to print the row and column names on the heatmap.

```
makeHeatmap(mat = comp_run$thresh_sigma, ## get matrix from plainER output
            title = "Thresholded Sigma Matrix",  ## plot title
            cluster = T, ## boolean for clustering
            names = T) ## boolean for displaying column/row names
```



Clusters

We use the `readER()` function, as mentioned above.

```
comp_read <- readER(x, comp_run) ## pass in parseRun results (after reading them in)
```

`readER()` lists the features in a cluster by their column indices in the `x` matrix. To convert these into the feature names (assuming your `x` had column names), we use the function `indName()`. This function takes in indices, the column names of your `x`, and a boolean flag indicating whether to convert from name to index (`T`) or index to name (`F`).

For example, if we wanted to see the first 10 pure and mixed variables, we would do the following:

```
indName(feats = comp_read$pure_vars[1:10], all_names = colnames(x), to_ind = F)
indName(feats = comp_read$mix_vars[1:10], all_names = colnames(x), to_ind = F)
```

Using `plainER()`

We recommend using a `.yaml` file for configuration and running Essential Regression using `parseRun()`. But for those that want to, below is a tutorial of `plainER()`

The function `plainER()` takes solely numerical data. If your response is categorical or ordinal, please take the time to transform it into a continuous variable. The `EssReg` package provides the `toCont()` function to assist in the transformation. `toCont()` has two arguments, `y` (the categorical response vector) and `order` (an optional argument that provides the levels of `y` in vector form). As an example, if you had injury severity as a response, then you could supply to `order` the vector `c("Mild", "Moderate", "Severe")`. The function will return a list including the categorical and continuous versions of `y` as well as the mapping performed.

Function Arguments Below are the function arguments:

- `y`: $\mathbf{Y}$, a vector $[n]$. REQUIRED
- `x`: $\mathbf{X}$, a data matrix $[n \times p]$. REQUIRED
- `delta`: δ, a numerical constant used for identifying latent cluster membership. We can specify either a single value or a range over which to search for the optimal value. REQUIRED
- `sigma`: $\hat{\Sigma}$, the sample covariance matrix calculated from $\mathbf{X}$ $[p \times p]$. REQUIRED (can be `NULL`)
- `thresh_fdr`: The false discovery rate threshold on $\hat{\Sigma}$. We use this value as the upper limit for the p-values of the entries of `sigma` - if a given p-value is less than `thresh_fdr`, the entry is set to 0 in `sigma`. DEFAULT = `0.2`
- `lambda`: λ, a numerical constant used in the LOVE algorithm for identifying latent cluster membership. DEFAULT = `0.1`
- `alpha_level`: The value used for confidence intervals when estimating the coefficients for the latent factor regression within Essential Regression. DEFAULT = `0.05`
- `rep_cv`: An integer indicating the number of replicates to perform when cross-validating to find δ. DEFAULT = `50`
- `out_path`: A string path to the directory in which to save output. OPTIONAL

Running `plainER()`

Now that we understand the function specification, we can run Essential Regression. When supplying the data to the function, make sure you are supplying matrices rather than paths. `plainER()` does not read files, so you must load the data before using the function. You must make sure to read in the data correctly; the example data has row names, so we use `row.names = 1` to indicate that the names are contained in the first column.

Also, if providing a vector of values for `delta`, we recommend using a sequence of the following format: `seq(start_val, end_val, step_size)`. For example, one may want to try a sequence from 0.01 to 0.1

stepping by 0.01 (written out, this is the sequence `(0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1)`. This can be done with: `seq(0.01, 0.1, 0.01)`.

```
x <- read.csv("path/to/x/.csv", row.names = 1) ## change path
y <- read.csv("path/to/y/.csv", row.names = 1) ## change path
initial_run <- plainER(y = y,
                       x = x,
                       sigma = cor(x),
                       thresh_fdr = 0.2,
                       delta = seq(0.01, 0.1, 0.01),
                       lambda = 0.5,
                       alpha_level = 0.05,
                       rep_cv = 50,
                       out_path = "temp/")
```

Notice that we have chosen to specify values for `sigma`, `thresh_fdr`, `lambda`, `alpha_level`, `rep_cv`, and `out_path`. Since we specified the default values for `sigma`, `thresh_fdr`, `alpha_level`. and `rep_cv`, we could get the exact same results using this code which omits the arguments from the function call:

```
initial_run <- plainER(y = y,
                       x = x,
                       delta = seq(0.01, 0.1, 0.01),
                       lambda = 0.5,
                       out_path = "temp/")
```