An Introduction to the lassoenet Package

Mokyo Zhou (mz37@st-andrews.ac.uk)

2018-04-21

Introduction

This vignette introduces the *lassoenet* package. I will go over the main user-facing functions and discuss the input arguments as well as the returns of the functions.

The *lassoenet* package provides many useful functions that can be used to fit different types of penalized linear regressions. More specifically, the Lasso (Tibshirani,1996), the Elastic Net (Zou & Hastie,2005) and the Adaptive Lasso (Zou, 2005) are all supported under the current version. The package puts great emphasis on flexibility and thus, it offers three different modes of use:

- Common Mode: contains main functions that can be directly implemented
- Interactive Mode: offers an interactive function that is used to guide users to achieve useful results
- Simulation Mode: provides users with tools that can be used to conduct simulation studies

1.Common Mode

For this mode, the *lassoenet* package offers three main user-facing functions for the fitting of penalised linear models. The *lassoenet* package calls suitable functions from the *glmnet* package (Friedman et al., 2014) and further develop on these foreign functions to create more user-friendly algorithms. I focus on adaptivity and thus, the package also introduces a great variation of fitting options to maximise performance. I will use the QuickStartExample data from the *glmnet* package for demonstration. Note, we assume the reader has basic understanding in how the Lasso (Tibshirani,1996), the Elastic Net (Zou & Hastie,2005) and the Adpative Lasso (Zou, 2005) work.

The QuickStartExample data consist of 100 data points collected on 20 numeric predictors. We will regress the response ,**y**, on all the predictors.

In this section, I will illustrate the fitting of the Lasso, the Elastic Net and the Adatpive Lasso with functions `prediction_Lasso`, `prediction_ElasticNet` and `Adlasso`, respectively.

1.1 `prediction_Lasso`

This function is mainly used to fit the Lasso models.

```
library(lassoenet)
library(glmnet)
data(QuickStartExample)
#Note we can access data in the following way:
data <- data.frame(y, x)
```

A basic use of the function will be:

```
result <- prediction_Lasso(data = data, x.indices = seq(2, 21),
                           response = 1, err.curves = 0, splits = 0,
                           type.lambda="lambda.min", interactive = FALSE, parallel = FALSE)
result
#> $seed
#> [1] 1234567
```

```
#>
#> $`best lambda`
#> [1] 0.07569327
#>
#> $`prediction error`
#> [1] 1.051569
#>
#> $prediciton_lower
#> [1] 0.8319358
#>
#> $prediction_upper
#> [1] 1.271203
```

To be more user friendly, users are no longer required to manually create and provide the standard `model.matrix` for the predictors. The `prediction_Lasso` function has an in-build conversion function that can convert any predictor matrix into the correct `model.matrix`. Users are only required to give the column numbers for the predictors through `x.indices` and the response through `response`. `err.curves` allows for a new alternative estimation of the tuning parameter λ (by averaging across multiple models) as opposed to the traditional set seed single Cross-Validation approach (which is used currently as `err.curves=0`). The `split` argument offers to split the dataset into a training and a testing set `c(splits,1-splits)`, where predictive performance can be measured more realistically. Here, `splits = 0` meaning we are using all the data without splitting.The `type.lambda` argument is used to specified the type of optimum λ, this can be either `lambda.min` or `lambda.1se` (Hastie & Qian, 2014). Note that if `err,curves != 0` the `type.lambda` argument will be ignored. When using this function please keep the argument `interactive = FALSE`. Multiple cores parallelisation is supported.

The return here consists of the seed that was used, the optimum λ that minimises the cross - validation errors, the cross -validation error corresponding to this optimum λ and the 95% confidence interval for the CV error(I calculate it by obtaining the standard error for this CV score (Hastie & Qian, 2014) and assuming normality). With these outputs users can reproduce the optimum solution. To find out more details of the arguments and the return, please visit the help page of the function `prediction_Lasso`. Some examples:

1.Keep everything the same as above and use 50 error curves.

```
result <- prediction_Lasso(data = data, x.indices = seq(2,21),
                           response = 1, err.curves = 50, splits = 0, parallel = FALSE)
```

```
result
#> $`number of err.curves`
#> [1] 50
#>
#> $`best lambda`
#> [1] 0.07569327
#>
#> $`prediction error`
#> [1] 1.057509
#>
#> $prediciton_lower
#> [1] 0.9930574
#>
#> $prediction_upper
#> [1] 1.132756
```

This returns the number of error curves used, the optimum λ that has the lowest averaged mean-squared

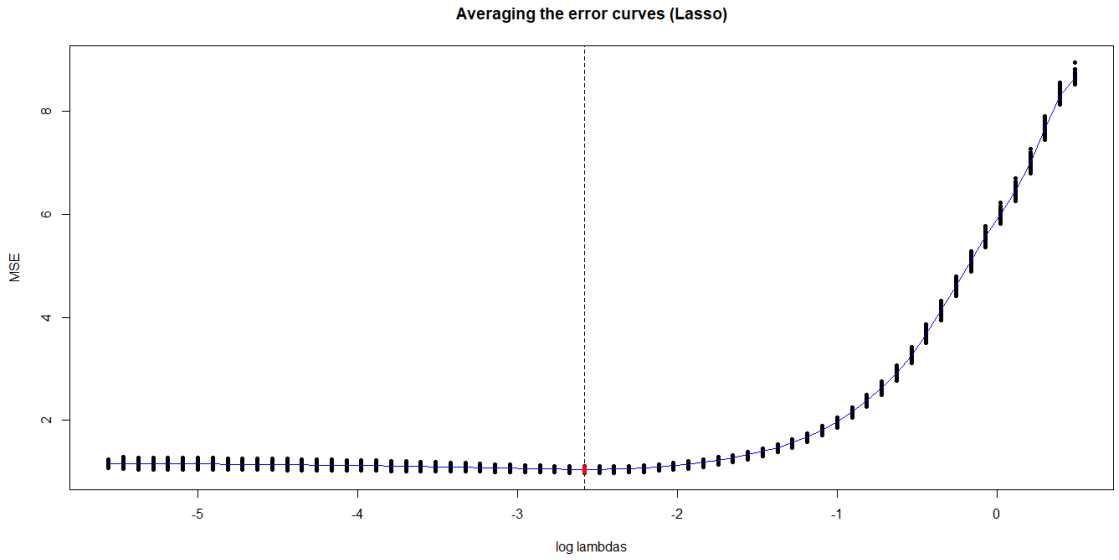


Figure 1: Error curves for the Lasso fitting process. Please note for the ease of visulisation, the error curves are plotted with the dot style, the "curves" are on top of each other. The blue line is the average across all the error curves. The red point is the optimum λ, the λ that has the lowest averaged error curves value.

error (lowest averaged error curves score), the averaged mean-squared error corresponding to the optimum λ and the 95% confidence interval for the averaged mean-squared error. Note the prediction error here is calculated as the lowest averaged error curves value whereas in the last example `err.curves=0` the prediction error is the (single) lowest 10-fold cross-validation score. The graph Fig 1 will also be outputted where the error curves and the prediction error can be examined.

---

2. Keep everything the same as above and splitting the dataset into a training set (80%) and a test set (20%) with lambda.1se as the optimum λ type

```
result <- prediction_Lasso(data = data, x.indices = seq(2,21),
                           response = 1, err.curves = 0, splits = 0.8,
                           type.lambda = "lambda.1se", parallel = FALSE)
result
#> $seed
#> [1] 1234567
#>
#> $`best lambda`
#> [1] 0.1505139
#>
#> $`prediction error`
#> [1] 0.8666735
#>
#> $`rooted prediciton error`
#> [1] 0.930953
#>
#> $`absolute prediction error`
#> [1] 0.8068976
```

This return the seed that was used, the optimum λ that minimizes the cross-validation error on the training set, the out of sample mean-squared error, the rooted mean-squared error and the mean absolute error on the

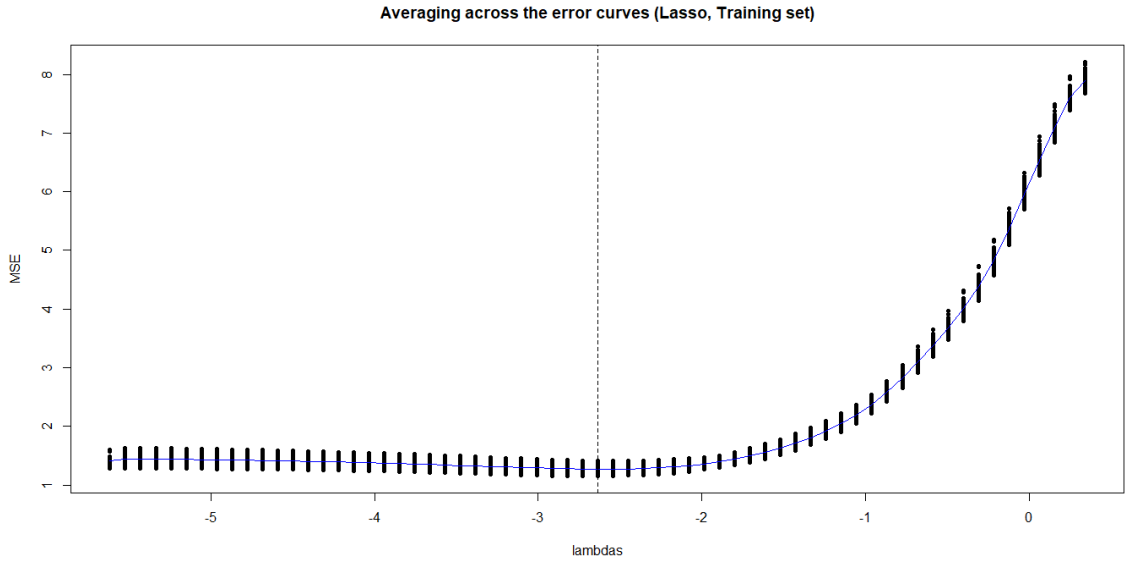


Figure 2: Error curves for the Lasso fitting process on the training set. Please note for the ease of visulisation, the error curves are plotted with the dot style, the "curves" are on top of each other. The blue line is the average across all the error curves. The interception between the blue line and the dash line is the optimum λ, the λ that has the lowest averaged error curves value on the training set.

test set. Note the different in the values of the `best lambda` from this example and the last example is due to the `type.lambda` argument.

---

3. 0.8 / 0.2 split, with 60 error curves with parallel (2 cores)

```
cl <- parallel::makeCluster(2)
doParallel::registerDoParallel(cl)
result <- prediction_Lasso(data = data, x.indices = seq(2,21),
                           response = 1, err.curves = 60, splits = 0.8, parallel = TRUE)
```

```
result
#> $`number of err.curves`
#> [1] 60
#>
#> $`best lambda`
#> [1] 0.07847806
#>
#> $`prediction error`
#> [1] 0.6265018
#>
#> $`rooted prediciton error`
#> [1] 0.7915187
#>
#> $`absolute prediction error`
#> [1] 0.6710464
```

This returns the number of error curves fitted on the training set, the optimum λ that has the lowest average mean-squared error on the training set. Out of sample mean-squared error, rooted mean-squared error and the mean absolute error on the test set. The graph figure 2 will also be outputted

1.2 prediction_ElasticNet

This function is similar to `prediction_Lasso` but with the main aim of fitting the Elastic Net models. Not only does this function contain all the key features that `prediction_Lasso` has, it also offers to find the optimum pair (α, λ) through an extensive 2-dimensional cross-validation process. In brief here, for each α, a 1-dimensional cross-validation over a range of λs is carried out, the local optimum pair is the pair (α, λ) that has the lowest CV error. After all the local optimum pairs and the corresponding CV errors have been collected, the global optimum pair is the pair that has the lowest global CV error. This is how the `set.seed` approach is carried out. For details of the 2-dimensional cross-validation repeated error curves approach, again please visit the help page for `prediction_ElasticNet`. We show some examples here:

---

1. non-split, no error curves, step.size =0.2, using `lambda.min` as optimum λ type

```
result <- prediction_ElasticNet(data = data, x.indices = seq(2,21),
                                 response = 1, err.curves = 0 ,splits = 0,
                                 step.size =0.2, type.lambda="lambda.min")
```

The only extra argument here is step.size, which specifies the step size of the [0,1] α grid. The outputs from this function is very similar to the `prediction_Lasso` function with the exception that an optimum pair of (α, λ) will be outputted instead of just the optimum λ.

---

2. no error curves, 0.8/0.2 split, step.size=0.1, using lambda.1se as the optimum λ type

```
result <- prediction_ElasticNet(data = data, x.indices = seq(2,21),
                                 response = 1, err.curves = 0, splits = 0.8,
                                 step.size = 0.1, type.lambda = "lambda.1se")
```

---

3. 100 error curves, 0.8/0.2 split, step.size=0.2, with parallel (2 cores)

```
cl <- parallel::makeCluster(2)
doParallel::registerDoParallel(cl)
result <- prediction_ElasticNet(data = data, x.indices = seq(2,21),
                                 response = 1, err.curves = 100, splits = 0.8,
                                 step.size = 0.2, parallel = TRUE)
```

1.3 Adlasso

This is the main function for fitting the Adaptive Lasso method. To be more convenient, this function provides a direct fit of the Adaptive Lasso model as opposed to the traditional manual two-stage fitting. The function supports two types of weight construction; "OLS" or "ridge". This means either a `lm()` or a `cv.glment(..., alpha = 0)` object will be created and the coefficients from which will be used to create the weights. A user definable γ grid is available to suit any particular needs. Furthermore, this function also supports the err.curves approach. When "OLS" is chosen for constructing the weights, the err.curves approach would involve a 2D cross-validation repeated fitting process.However, when "ridge" is chosen to construct the weights, the function will run a 3D cross-validation repeated curves algorithm to determine the optimum triple (γ, λ, `ridge.lambda`). For more details please see the help page for `Adlasso`.

In contrast to the `prediction_Lasoo` and `prediction_ElasticNet`, the `Adlasso` function focuses on making valid inference. Thus, the function would use the whole dataset to improve inference as opposed to splitting the dataset into a training and a testing set.

Some examples:

---

1. no error curves, using "OLS" coefficients to construct weights, γ grid = c(0.5,1,2), optimum type is lambda.min

```
result <- Adlasso(data = data, x.indices = seq(2,21), response = 1, err.curves = 0 ,
                  weight.method = "OLS", gamma.seq = c(0.5,1,2), type.lambda="lambda.min",
                  B.rep = 500, significance = 0.05)

result
#> $seed
#> [1] 1234567
#>
#> $`best gamma`
#> [1] 2
#>
#> $`best lambda`
#> [1] 3.140011
#>
#> $`prediction error`
#> [1] 0.9622348
#>
#> $prediciton_lower
#> [1] 0.7577473
#>
#> $prediction_upper
#> [1] 1.166722
#>
#> $`%null deviance explained`
#> [1] 0.9075424
#>
#> $CIs
#>           Estimates  0.025  0.975 proportion of non-zero estimates
#> intercept     0.132 -0.040  0.357                             1.000
#> X1            1.388  1.193  1.552                             1.000
#> X2            0.000 -0.137  0.157                             0.252
#> X3            0.775  0.607  0.958                             1.000
#> X4            0.000 -0.159  0.145                             0.294
#> X5           -0.906 -1.093 -0.701                             1.000
#> X6            0.592  0.420  0.790                             1.000
#> X7            0.000 -0.140  0.144                             0.236
#> X8            0.388  0.209  0.716                             0.984
#> X9            0.000 -0.108  0.164                             0.234
#> X10           0.000 -0.147  0.151                             0.280
#> X11           0.216  0.046  0.432                             0.754
#> X12           0.000 -0.147  0.148                             0.250
#> X13           0.000 -0.176  0.153                             0.250
#> X14          -1.120 -1.297 -0.918                             1.000
#> X15           0.000 -0.196  0.188                             0.268
#> X16           0.000 -0.168  0.130                             0.296
#> X17           0.000 -0.162  0.179                             0.252
#> X18           0.000 -0.093  0.167                             0.268
#> X19           0.000 -0.127  0.105                             0.252
```

```
#> X20          -1.147 -1.368 -0.925                                  1.000
```

The function returns the seed that was used, the optimum (γ, λ) pair that has the lowest cross-validation error, the cross-validation error corresponds to the optimum (γ, λ) and its 95% confidence interval. The output also gives the precentage of deviance explained, which according to the glmnet() function (Hastie & Qian, 2014), can be used as a form of goodness of fit. Furthermore, the package also provides the $100(1-\alpha)\%$ confidence intervals for the model parameters. These are generated by using a residual bootstrap algorithm within the `lassoenet` package. Users can define the number of bootstraps and the significance level with the arguments `B.rep` and `significance`, respectively. The table also gives information about the proportion of non-zero estimates, for each parameter, out of `B.rep` bootstrapping. *Note that the example is only for demonstration purpose and thus, for a thorough analysis, the user is recommended to expand the γ grid to explore larger γ values.*

---

2. no error curves, using "ridge" coefficients to construct weights, gamma.seq = seq(1,10), optimum λ type is lambda.1se

```
result <- Adlasso(data = data, x.indices = seq(2,21), response = 1, err.curves = 0,
                  weight.method = "ridge", gamma.seq = seq(1,10), type.lambda = "lambda.1se")
```

---

3. 80 error curves, weight.method = "ridge", gamma.seq=c(0,0.5,1,2,2.5),with parallel (2 cores)

```
cl <- parallel::makeCluster(2)
doParallel::registerDoParallel(cl)
result <- Adlasso(data = data, x.indices = seq(2,21), response = 1, err.curves = 80,
                  weight.method = "ridge", gamma.seq = c(0,0.5,1,2,2.5), parallel = TRUE)
```

Note when weight.method = "ridge", e.g. example 2 and 3, in addition to the outputs in example 1, the function will also return the optimum λ for the ridge regression. With all these outputs, users can freely reproduce the results.

---

2. Interactive Mode

The `lassoenet` package offers an uniquely designed, first of its kind interactive algorithm that aims to guide interested users to obtain better suited penalized models to their own datasets. The package achieves this aim by interacting with the users through a series of questions and suggestions. I try to equip each question with some useful and practical hints that would hopefully steer the user into a more appropriate modelling direction and spark research interest. The function has two main emphases: prediction accuracy and making inference. For the prediction accuracy, the function mainly support the Lasso and the Elastic Net (two of the main functions from the common mode). On the other hand, the function would mainly uses the Adaptive Lasso model (support both "OLS" and "ridge" weightings) for inference purposes.

We will use the `interactiver()` function to conduct interactive studies

```
library(lassoenet)
library(glmnet)
data(QuickStartExample)
row.samples <- sample(1:100,250,TRUE)#enlarging the sample size and access the data
data <- data.frame(y,x)[row.samples,]
result <- interactiver(data = data, parallel = FALSE, ncores = NULL)
```

Unfortunately, R markdown does not support this type of interactive function. I will screenshot the process for demonstration:

```
> result <- interactiver(data = data)
Thank you for using our interactive process. This process
will automatically generate the desired outputs based on your input
arguments. Therefore, please read the questions carefully and ONLY
enter your inputs in the format shown. Thank you :)

1. please be sure to process your data appropreiately before
using this package (e.g. missing values, outliers and assumptions checking)
as dirty data would heavily influence the result quality.

2. Furthermore, the dataset must be either a rectangular
data.farme or an as.matrix object with number of rows >> than the number
of columns. Are you happy to precess? Yes or NoYes
Yes

3. Which is the focus inference or prediction? please only
enter inference or prediction?                                    inference
inference


4. what is your response variable's column index.  e.g. 1              1
response index = 1


5. what are the column indices for the covariates.  e.g.
c(seq(2,4),5,seq(6,10))                                            seq(2,21)
covariate columns = 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

6. The tunning process is random. One way to stabilise
the tunning process is by averaging across multiple error curves. This might
give better solutions but more time consuming.Stabilise Yes or No?
(More info from main functions e.g. prediction_Lasso)Yes
Averaging across error curves = Yes

7. please specify the number of error curves. e.g. 100                  20
number of error curves = 20

8. The process will run the adaptive lasso for inference. The
adaptive lasso reduces bias by allowing each covariate to have a different
penalty. Thus, useful covariates can have a smaller penalty while irrelevant
coefficents will be penalised more heavil

9. Use either OLS or ridge coefficients to constuct a penalty
vector to allow for different penalisation. If collinearity is a problem,
then ridge regressive might be a better starting weight.The Following is
the VIFs based on the linear model:

      X1       X2       X3       X4       X5       X6       X7       X8
1.345685 1.345104 1.334657 1.477232 1.331639 1.209577 1.331244 1.394202
      X9      X10      X11      X12      X13      X14      X15      X16
1.238869 1.239164 1.422750 1.567163 1.237892 1.610801 1.678549 1.175174
     X17      X18      X19      X20
1.340282 1.278213 1.218430 1.421076

10. Any VIFs above 5 or GVIF^(1/(2*Df)) > 2 indicate collinearity.
But keep in mind that there are many more ways in which collinearity can be
checked and we are only showing 1 of such ways

11. What cofficients should be used for the construction
of the penalty vector OLS or ridge? Base your answer on the outputs above
as well as any other detection teachniques and personal judgementOLS
Type of coefficents for inital weight construction = OLS
```

```
12. Please define the gamma sequence which you would like
to try. Note this gamma sequence can have range from 0 to positive infinite.
Please enter in the form of e.g. seq(0,5) or c(0,0.5, 1,...)c(0.5,1,2)
The gamma sequence to use = 0.5 1 2

13. Please define the number of residual bootstrappings
that you would like to do. Please note that the more bootstrappings you do,
the less uncertain will your result be but more time consuming60
Number of residual bootstrappings to do = 60

14. Please define the significance level of the residual
bootstrap, e.g. 0.05 for a 95% confidence interval0.05
The significance level for the residual bootstrap = 0.05
```

The above is one of the many different fitting paths that are available to the users. As from the demonstration, the function tries to provide useful information about the method and offer practical hints to guide the user to a more suitable direction. In this particular example, the Adaptive Lasso has been employed to extract inference from the dataset. The weights were built from the coefficients from a `lm()` object and the error curves approach has also been called. The confidence intervals were generated from 60 residual boos trappings and the significance level has been set to 0.05. The returns from this call are largely the same as from the `Adlasso` function.

For interactive prediction (omitted here), on the other hand, both `prediction_Lasso` and `prediction_ElasticNet` functions would be called. The function would offer similar helpful suggestions and hints that are more relevant to the Lasso and the Elastic Net methods. The function would also provide an automatic predictive performance comparison between the Lasso and the Elastic Net based on a prediction metric chosen by the user.

This `interactiver` function also fully supports multiple cores parallelisation, which users can activate by setting `parallel = TRUE` and specify the `ncores` argument.

3. Simulation Mode

This mode is designed for users who are more interested in learning the theoretical properties behind the methods Lasso, Elastic Net, ridge and the Adaptive Lasso. The package offers a very powerful data simulation function, which can be used to generate various types of data based on the user's preference. The function `simulation.generation.data` does this task (single simulated dataset):

```
result <- simulation.generation.data(n = 100, coeff = c(1,2,3,4), matrix.option = 1,
                                     collinear = 0.3, sig = 2, split.prop = 0.8, option = 1)
```

Here, `n` is the number of observations, `coeff` contains the true coefficients. `Matrix.option = 1`, generates the data from a multivariate normal distribution with a compound symmetric (exchangeable) correlation matrix. `Matrix.option =2`, generates the data based on an auto regressive correlation matrix. The `collinear` argument specifies the collinearity level within the correlation matrix. Model error can be defined by the `sig` argument. The `split.prop` determines the training proportion (test proportion will be 1 - training proportion). `option =1`, the function will do the data splitting. `option = 2`, the `split.prop` argument will be ignored, and return the whole simulation dataset.

Users can wrap this data generating function into their own simulation functions to conduct investigations. Or if the user prefers, the package also provides simulation functions that can be used to carry out the full simulation study.

The functions `simulation.collinear` and `simulation.adlasso` do such task:

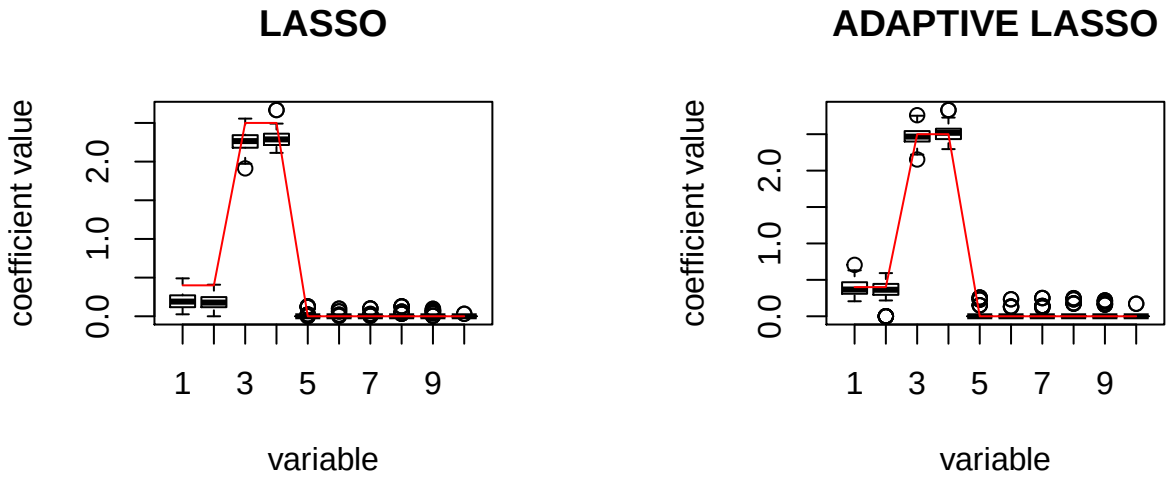
```
cl <- parallel::makeCluster(2)
doParallel::registerDoParallel(cl)
result <- simulation.collinear(n.resample = 20, n = 200, coeff = c(10,8,0,0,12,0,0,0,0,0),
```

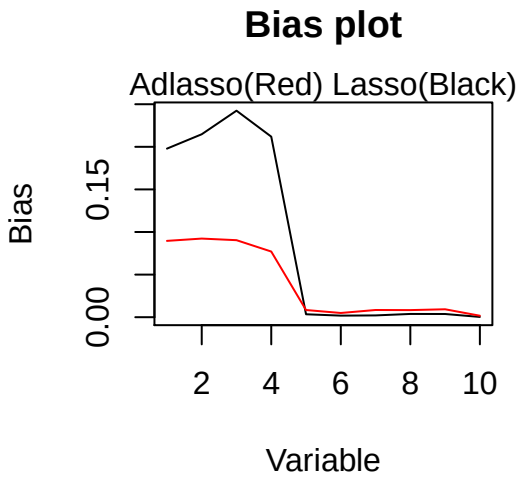
```
                                matrix.option = 2, collinear = 0.2, sig = 3, split.prop = 0.6,
                                parallel = TRUE, step.alpha = 0.2, option= 1)

result$final.table
#>                                        Lasso Elastic Net      Ridge
#> Average External MSE              10.0763606  10.1126371 11.7918007
#> Standard Error                     0.3789298   0.4076022  0.5206566
#> Average Model selection Accuracy   0.9000000   0.7000000         NA
#> Average alpha                      1.0000000   0.7200000  0.0000000
```

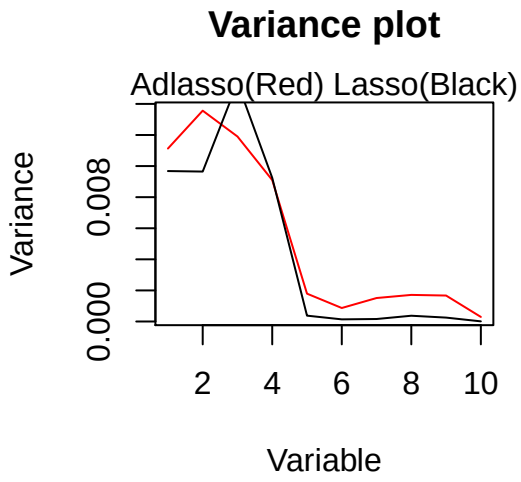
This function wraps around the `simulation.generation.data` function and compares the predictive performance between the methods Lasso, the Elastic Net and the ridge regression. The n.resample argument specifies the number of simulations to do (e.g. how many datasets should be generated from the `simulation.generateion.data` function). The conditions within the simulated datasets can be defined through the rest of the arguments. Finally, users will have to define the step size of the α grid too, as the Elastic Net will also be run. The function returns four elements, with the first being a summary table of the simulation process, please visit the help page of `simulation.collinear`. In case the users would like to conduct further investigation, the function also returns the full simulation matrices for the three methods (the last three elements from the return).

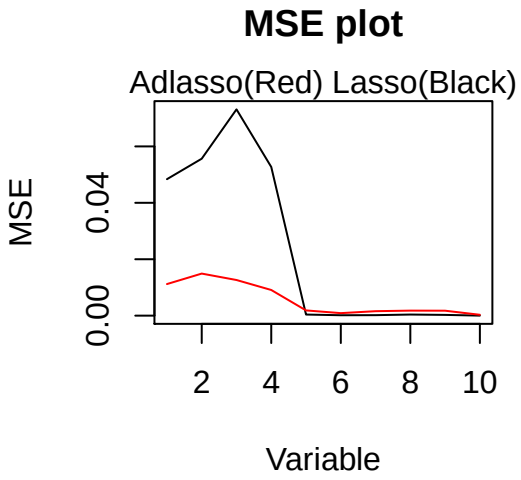
On the other hand, the function `simulation.adlasso` is mainly to study the inference performance between the Lasso and the Adaptive Lasso

```
cl <- parallel::makeCluster(2)
doParallel::registerDoParallel(cl)
result <- simulation.adlasso(n.resample = 200, n = 400, coeff = c(0.4,0.4,2.5,2.5,rep(0,6)),
                             matrix.option = 1, collinear = 0.2, sig = 2, parallel = TRUE,
                             option=2)
```










```
result$final.table
#>                                  Lasso Adaptive Lasso
#> average incorrect classifications 0.305          0.260
#> model accuracy                    0.740          0.785
```

This function has similar arguments as the function `simulation.collinear`. Please note, I encourage the users to keep the argument option = 2 fixed (using the full simulated datasets), as out of sample prediction is not a focus here. I also need to point out that, here, the weights for the Adaptive Lasso have been constructed based on a `cv.glmnet(..., alpha = 1)` fit. The function returns three elements, with the first being a summary table of the inference performance between the Adaptive Lasso and the Lasso. To give more insight into the performance, the function also provides visualizations of bias, variance and MSE. To allow for further investigation, both simulation matrices for the Adaptive Lasso and the Lasso are also returned.

References

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. J. R. Statist. Soc. B, 58, 267-288.

Zou, H. (2006). The adaptive lasso and its oracle properties. Journal of the American Statistical Association, 101(476), pp.1418-1429.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), pp.301-320.

Friedman,J.,Hastie,T.,Tibshirani,R.,Simon,N.,Narasimhan,B. and Qian,J. (2014). glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models. Available online at: [https://cran.r-project.org/web/packages/glmnet/index.html].

Hastie,T., and Qian,J. (2014). Glmnet Vignette. Available online at [https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html].