Regularization and Variable Selection: Ridge Regression, LASSO, and Elastic Net

Günter J. Hitsch

February 15, 2017

glmnet package

For more information on the `glmnet` package please consult the corresponding vignette.

Data

We simulate a data set with "true" and "false" inputs. The true inputs affect the outcome variable (with a coefficient $\beta_k = 1$), the false inputs have no impact on the output. The inputs, x_k, are uniformly distributed on $[0, 1]$, and the error term is drawn from a normal distribution centered at 0.

```
N_true_inputs  = 5       # Number of true inputs
N_false_inputs = 20      # Number of false inputs
n_obs          = 1000    # Number of observations
error_sd       = 1       # Standard deviation of the error term

# Total number of inputs (not including the intercept)
p = N_true_inputs + N_false_inputs

# Paramter vector: True inputs have coefficient = 1.0
beta = c(rep(1.0, N_true_inputs), rep(0.0, N_false_inputs))

# Simulate the data: Note that X is a matrix, not a data.table
set.seed(963)
X       = matrix(runif(n_obs*p), nrow = n_obs, ncol = p)
epsilon = rnorm(n_obs, sd = error_sd)
y       = X %*% beta + epsilon

# Pack the data into a data.table
DT = as.data.table(X)
DT[, y := y]
new_names_true  = paste0("x_true_", 1:N_true_inputs)
new_names_false = paste0("x_false_", 1:N_false_inputs)
setnames(DT, names(DT), c(new_names_true, new_names_false, "y"))
```

Linear regression

```
fit_OLS = lm(y ~ ., data = DT)
summary_OLS = summary(fit_OLS)

# Collect results
results = data.table(input   = rownames(summary_OLS$coefficients),
                     est_OLS = summary_OLS$coefficients[, 1],
                     p_OLS   = summary_OLS$coefficients[, 4])

summary(fit_OLS)
```

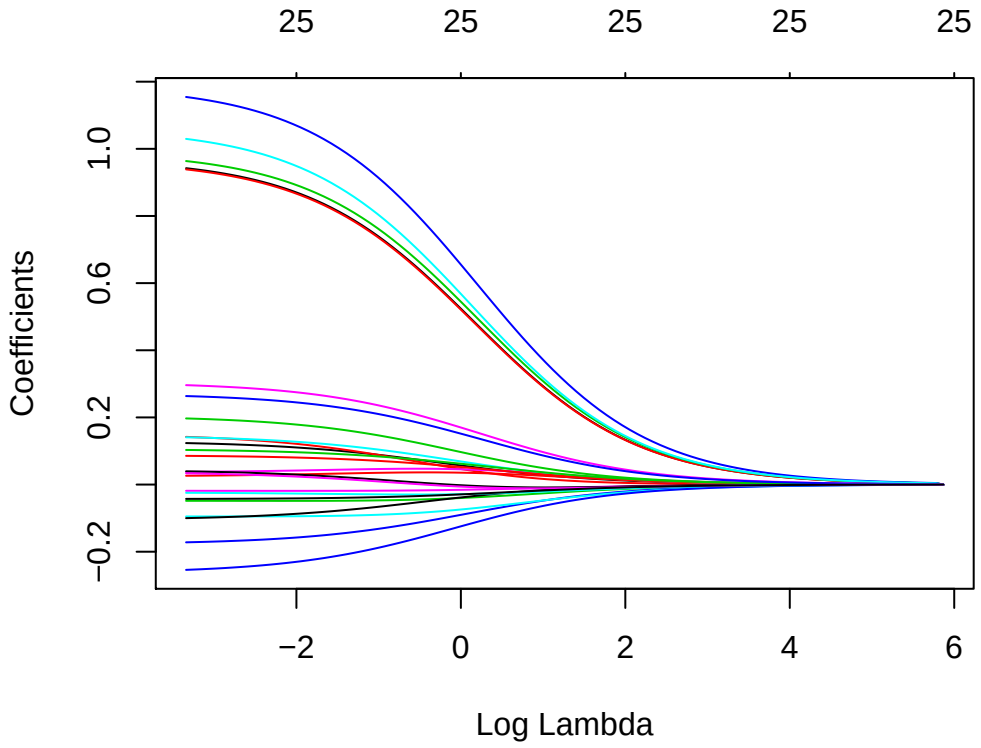
Ridge regression

First, a note on how to use `glmnet`. The general formula is

```
fit = glmnet(x = X, y = y, alpha = ...)
```

`X` is a matrix with inputs as columns, and `y` is the output vector. `alpha` is the elastic-net parameter. To estimate a ridge regression, we set `alpha = 0`:

```
fit_ridge = glmnet(x = X, y = y, alpha = 0.0)
plot(fit_ridge, xvar = "lambda")
```



Typically your data will be a data.table (or data frame) `DT`. To create the `X` matrix and the `y` use:

```
X = model.matrix(y_var ~ 0 + x_var_1 + x_var_2 + ..., data = DT)
y = DT$y_var
```

Note how to use the `model.matrix` formula: You specify an R formula corresponding to the regression that you would like to estimate. In addition you need to add `0 +` to the right-hand side of the formula to remove the intercept. The intercept is automatically supplied by `glmnet`. In our example:

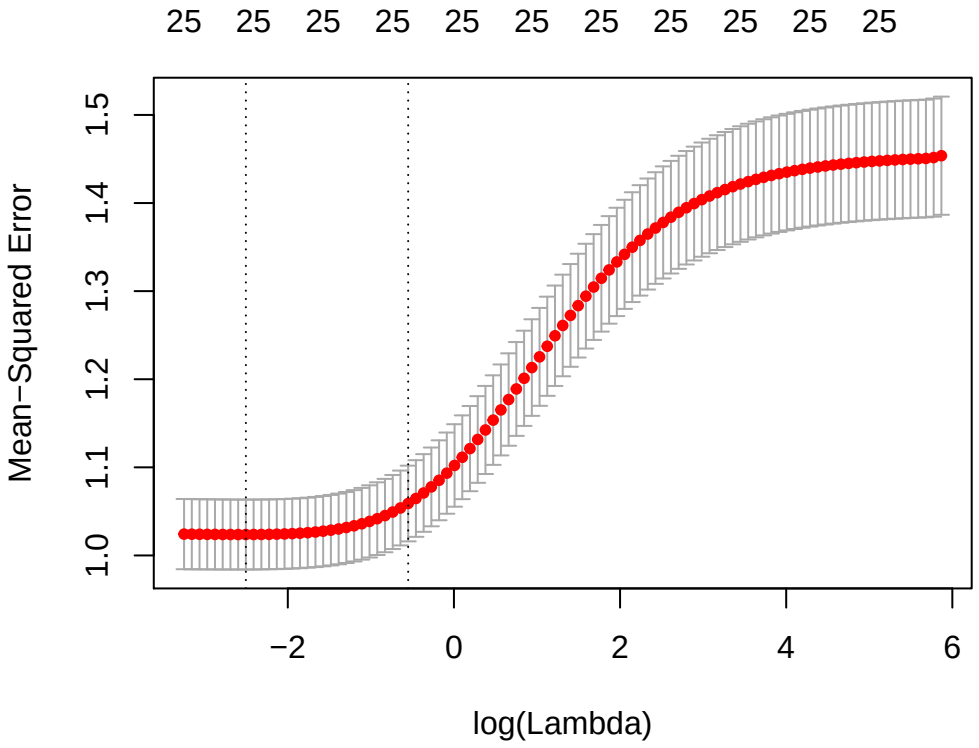
```
X_ = model.matrix(y ~ 0 + ., data = DT)
y_ = DT$y

sum(abs(X_ - X))      # X_ and X, y_ and y are numerically identical
sum(abs(y_ - y))
```

Now we use cross-validation to find the tuning parameter that provides the best out-of-sample fit. `cv.glmnet` provides a range of model estimates for different tuning parameters, λ. To find the coefficients for the λ value with the lowest cross-validation error we use the `s = "lambda.min"` option. Note that `"lambda.1se"` selects the tuning parameter for the most regularized model within one standard deviation of the minimum cross-validation error.

```
cv_ridge = cv.glmnet(x = X, y = y, alpha = 0.0)
cv_ridge$lambda.min
cv_ridge$lambda.1se
results[, est_ridge := coef(cv_ridge, s = "lambda.min")[,1]]
coef(cv_ridge, s = "lambda.min")
```

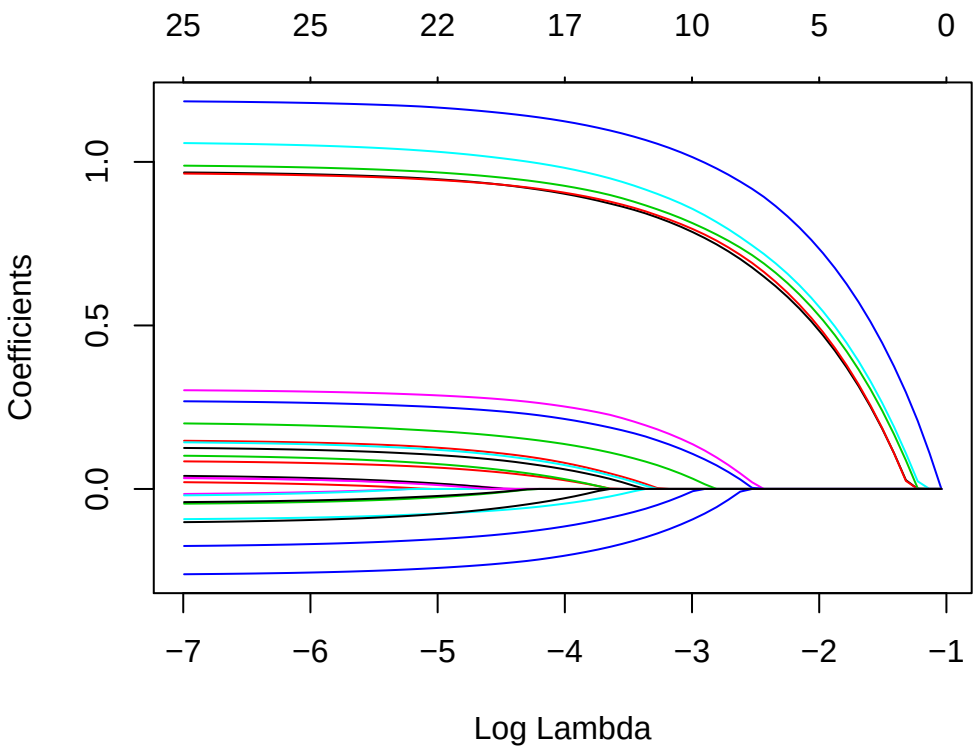
```
plot(cv_ridge)
```



LASSO

By default, `alpha = 1`, and hence the elastic-net parameter α need not be provided to `glmnet` when estimating a LASSO:

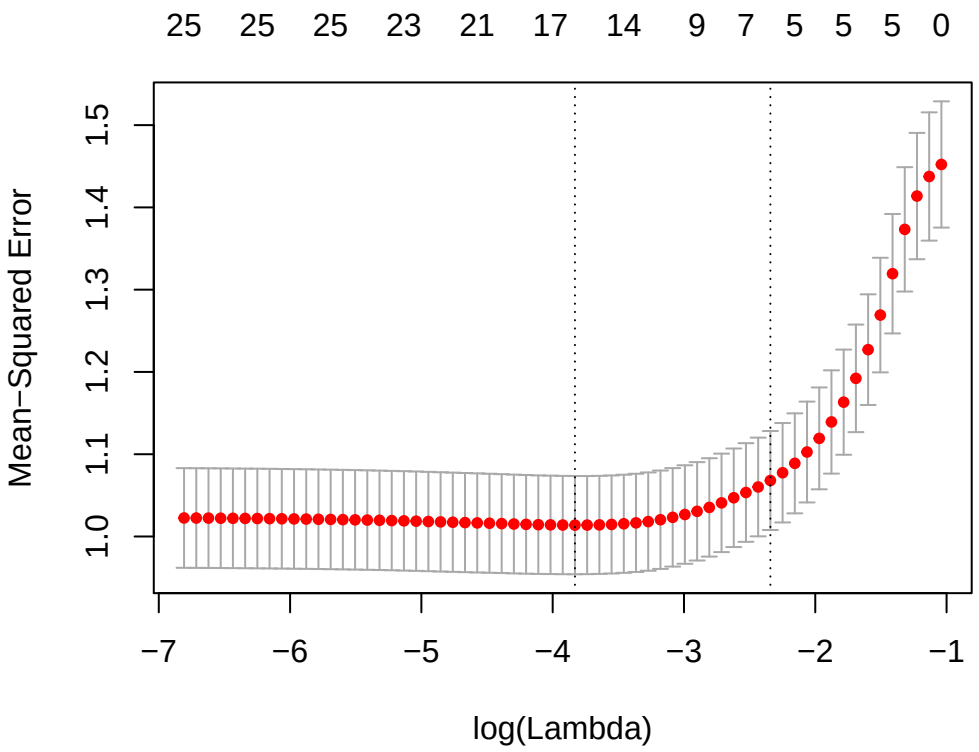
```
fit_LASSO = glmnet(x = X, y = y)
plot(fit_LASSO, xvar = "lambda")
```



Cross-validation:

```
cv_LASSO = cv.glmnet(x = X, y = y)
cv_LASSO$lambda.min
cv_LASSO$lambda.1se
results[, est_LASSO := coef(cv_LASSO, s = "lambda.min")[,1]]
results[, est_LASSO_1se := coef(cv_LASSO, s = "lambda.1se")[,1]]
coef(cv_LASSO, s = "lambda.min")
```

```
plot(cv_LASSO)
```



Out-of-sample prediction

We first simulate new inputs and a corresponding output.

```
n_new = 100000

set.seed(901)
X_new       = matrix(runif(n_new*p), nrow = n_new, ncol = p)
epsilon_new = rnorm(n_new, sd = error_sd)
y_new       = X_new %*% beta + epsilon_new

DT_new = as.data.table(X_new)
setnames(DT_new, names(DT_new), c(new_names_true, new_names_false))
```

Calculate the mean-squared-error of the predictions:

```
pred_y_OLS = predict(fit_OLS, newdata = DT_new)
mse_OLS    = mean((y_new - pred_y_OLS)^2)

pred_y_ridge = predict(cv_ridge, newx = X_new, s = "lambda.min")
mse_ridge    = mean((y_new - pred_y_ridge)^2)

pred_y_LASSO = predict(cv_LASSO, newx = X_new, s = "lambda.min")
mse_LASSO    = mean((y_new - pred_y_LASSO)^2)
```

A note on how to use the `model.matrix` function. Because `DT_new` does not contain the output that we would like to predict, the output must not be used in the model formula:

```
X_new_ = model.matrix(~ 0 + ., data = DT_new)
```

Generally, be **very careful when you specify the formula in `model.matrix`**. The resulting matrix needs to contain exactly the same variables as in the original regression, and the columns need to be in the correct

order, corresponding to the order of the variables in the matrix that was used to estimate the model! One common mistake *if* your data.table (data frame) contains an output:

```
DT_new_ = as.data.table(X_new)
DT_new_[, y := y_new]
setnames(DT_new_, names(DT_new_), c(new_names_true, new_names_false, "y"))
```

```
X_new_ = model.matrix(~ 0 + ., data = DT_new_)
ncol(X_new)
```

```
[1] 25
```

```
ncol(X_new_)
```

```
[1] 26
```

`X_new_` has one more column than `X_new`, because here the output was **not** specified on the left-hand side of the model formula. Hence, the output was added to the columns on the right-hand side. To create the correct matrix of inputs in this example we need to use:

```
X_new_ = model.matrix(y ~ 0 + ., data = DT_new_)
```

The regression results and MSE's:

```
kable(results, digits = 3)
```

input	est_OLS	p_OLS	est_ridge	est_LASSO	est_LASSO_1se
(Intercept)	-0.456	0.101	-0.256	-0.092	0.766
x_true_1	0.971	0.000	0.907	0.889	0.620
x_true_2	0.967	0.000	0.904	0.894	0.633
x_true_3	0.992	0.000	0.929	0.915	0.661
x_true_4	1.188	0.000	1.113	1.112	0.865
x_true_5	1.062	0.000	0.990	0.968	0.690
x_false_1	-0.019	0.866	-0.019	0.000	0.000
x_false_2	0.129	0.254	0.117	0.047	0.000
x_false_3	0.151	0.165	0.132	0.066	0.000
x_false_4	-0.048	0.671	-0.049	0.000	0.000
x_false_5	-0.178	0.107	-0.165	-0.103	0.000
x_false_6	-0.095	0.402	-0.095	-0.035	0.000
x_false_7	0.036	0.745	0.040	0.000	0.000
x_false_8	-0.105	0.335	-0.094	-0.015	0.000
x_false_9	0.024	0.828	0.028	0.000	0.000
x_false_10	0.204	0.069	0.188	0.126	0.000
x_false_11	-0.264	0.018	-0.242	-0.192	0.000
x_false_12	0.146	0.189	0.134	0.059	0.000
x_false_13	0.304	0.007	0.286	0.241	0.000
x_false_14	0.044	0.695	0.035	0.000	0.000
x_false_15	0.087	0.429	0.083	0.016	0.000
x_false_16	0.106	0.340	0.100	0.017	0.000
x_false_17	0.271	0.015	0.255	0.203	0.000
x_false_18	-0.023	0.831	-0.026	0.000	0.000
x_false_19	0.038	0.734	0.029	0.000	0.000
x_false_20	-0.043	0.694	-0.042	0.000	0.000

```r
cat(mse_OLS, mse_ridge, mse_LASSO, "\n")
```

```
1.034395 1.029136 1.014003
```

Elastic net: Tune the `alpha` parameter

We now manually tune the `alpha` parameter to estimate an elastic net (no such tuning function is directly provided by `glmnet`).

First, simulate a new data set. Here, the "true" inputs are uniformly distributed on $[-b, b]$, where b is specified using the `bounds` parameter below.

```
set.seed(963)

N_true_inputs  = 10      # Number of true inputs
N_false_inputs = 5       # Number of false inputs
bound          = 0.9     # Bound value for simulation of coefficients
n_obs          = 1000    # Number of observations
error_sd       = 1       # Standard deviation of the error term

# Total number of inputs (not including the intercept)
p = N_true_inputs + N_false_inputs

# Paramter vector
beta = c(runif(N_true_inputs, min = -bound, max = bound), rep(0.0, N_false_inputs))

# Simulate the data
X       = matrix(runif(n_obs*p), nrow = n_obs, ncol = p)
epsilon = rnorm(n_obs, sd = error_sd)
y       = X %*% beta + epsilon
```

Now we find the `alpha` value that provides the lowest cross-validation error. We thus estimate the elastic net over a grid of α values, $\alpha = 0, 0.01, \ldots, 0.99, 1$.

Note that we provide the fold numbers (`folds`) directly to `cv.glmnet`. This is important, so that we can compare the prediction error for the *exact same folds* across different `alpha` values.

```
set.seed(1999)

# Create the folds
folds = sample(1:10, n_obs, replace = TRUE)

# Output table
rmse_DT = data.table(alpha         = seq(0, 1, by = 0.01),
                     mean_cv_error = rep(0, 101))

# Calculate cross-validation error for different alpha values
for (i in 0:100) {
   cv_i = cv.glmnet(x = X, y = y, alpha = rmse_DT[i+1, alpha], foldid = folds)
   rmse_DT[i+1, mean_cv_error := min(cv_i$cvm)]
}

# Optimal alpha:
index_min = which.min(rmse_DT$mean_cv_error)
rmse_DT[index_min, alpha]
```

```
[1] 0.67
```

Experiment with the settings. You will find that a ridge regression will be favored if the true regression coefficients are small, and a LASSO will be favored if the coefficients are large. A LASSO will also be favored if there are many coefficients that are zero.