Articles - Model Selection Essentials in R

Penalized Regression Essentials: Ridge, Lasso & Elastic Net

kassambara | *11/03/2018* | *1550* | *Post a comment* | *Model Selection Essentials in R*

The standard linear model (or the ordinary least squares method) performs poorly in a situation, where you have a large multivariate data set containing a number of variables superior to the number of samples.

A better alternative is the **penalized regression** allowing to create a linear regression model that is penalized, for having too many variables in the model, by adding a constraint in the equation (James et al. 2014,P. Bruce and Bruce (2017)). This is also known as **shrinkage** or **regularization** methods.

The consequence of imposing this penalty, is to reduce (i.e. shrink) the coefficient values towards zero. This allows the less contributive variables to have a coefficient close to zero or equal zero.

Note that, the shrinkage requires the selection of a tuning parameter (lambda) that determines the amount of shrinkage.

In this chapter we'll describe the most commonly used penalized regression methods, including **ridge regression**, **lasso regression** and **elastic net regression**. We'll also provide practical examples in R.

Contents:

Shrinkage methods

Ridge regression

Ridge regression shrinks the regression coefficients, so that variables, with minor contribution to the outcome, have their coefficients close to zero.

The shrinkage of the coefficients is achieved by penalizing the regression model with a penalty term called **L2-norm**, which is the sum of the squared coefficients.

The amount of the penalty can be fine-tuned using a constant called lambda (λ). Selecting a good value for λ is critical.

When $\lambda = 0$, the penalty term has no effect, and ridge regression will produce the classical least square coefficients. However, as λ increases to infinite, the impact of the shrinkage penalty grows, and the ridge regression coefficients will get close zero.

Note that, in contrast to the ordinary least square regression, ridge regression is highly affected by the scale of the predictors. Therefore, it is better to standardize (i.e., scale) the predictors before applying the ridge regression (James et al. 2014), so that all the predictors are on the same scale.

The standardization of a predictor `x`, can be achieved using the formula `x' = x / sd(x)`, where sd(x) is the standard deviation of x. The consequence of this is that, all standardized predictors will have a standard deviation of one allowing the final fit to not depend on the scale on which the predictors are measured.

One important advantage of the ridge regression, is that it still performs well, compared to the ordinary least square method (Chapter @ref(linear-regression)), in a situation where you have a large multivariate data with the number of predictors (p) larger than the number of observations (n).

One disadvantage of the ridge regression is that, it will include all the predictors in the final model, unlike the stepwise regression methods (Chapter @ref(stepwise-regression)), which will generally select models that involve a reduced set of variables.

Ridge regression shrinks the coefficients towards zero, but it will not set any of them exactly to zero. The lasso regression is an alternative that overcomes this drawback.

Lasso regression

Lasso stands for Least Absolute Shrinkage and Selection Operator. It shrinks the regression coefficients toward zero by penalizing the regression model with a penalty term called **L1-norm**, which is the sum of the absolute coefficients.

In the case of lasso regression, the penalty has the effect of forcing some of the coefficient estimates, with a minor contribution to the model, to be exactly equal to zero. This means that, lasso can be also seen as an alternative to the subset selection methods for performing variable selection in order to reduce the complexity of the model.

As in ridge regression, selecting a good value of λ for the lasso is critical.

One obvious advantage of lasso regression over ridge regression, is that it produces simpler and more interpretable models that incorporate only a reduced set of the predictors. However, neither ridge regression nor the lasso will universally dominate the other.

Generally, lasso might perform better in a situation where some of the predictors have large coefficients, and the remaining predictors have very small coefficients.

Ridge regression will perform better when the outcome is a function of many predictors, all with coefficients of roughly equal size (James et al. 2014).

Cross-validation methods can be used for identifying which of these two techniques is better on a particular data set.

Elastic Net

Elastic Net produces a regression model that is penalized with both the **L1-norm** and **L2-norm**. The consequence of this is to effectively shrink coefficients (like in ridge regression) and to set some coefficients to zero (as in LASSO).

Loading required R packages

- `tidyverse` for easy data manipulation and visualization
- `caret` for easy machine learning workflow
- `glmnet`, for computing penalized regression

```
library(tidyverse)
library(caret)
library(glmnet)
```

Preparing the data

We'll use the `Boston` data set [in `MASS` package], introduced in Chapter @ref(regression-analysis), for predicting the median house value (`mdev`), in Boston Suburbs, based on multiple predictor variables.

We'll randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model). Make sure to set seed for reproducibility.

```
# Load the data
data("Boston", package = "MASS")
# Split the data into training and test set
set.seed(123)
training.samples <- Boston$medv %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- Boston[training.samples, ]
test.data <- Boston[-training.samples, ]
```

Computing penalized linear regression

Additional data preparation

You need to create two objects:

- `y` for storing the outcome variable
- `x` for holding the predictor variables. This should be created using the function `model.matrix()` allowing to automatically transform any qualitative variables (if any) into dummy variables (Chapter @ref(regression-with-categorical-variables)), which is important because glmnet() can only take numerical, quantitative inputs. After creating the model matrix, we remove the intercept component at index = 1.

```
# Predictor variables
x <- model.matrix(medv~., train.data)[,-1]
# Outcome variable
y <- train.data$medv
```

R functions

We'll use the R function `glmnet()` [glmnet package] for computing penalized linear regression models.

The simplified format is as follow:

```
glmnet(x, y, alpha = 1, lambda = NULL)
```

- `x`: matrix of predictor variables
- `y`: the response or outcome variable, which is a binary variable.
- `alpha`: the elasticnet mixing parameter. Allowed values include:
    - "1": for lasso regression

  - “0”: for ridge regression
  - a value between 0 and 1 (say 0.3) for elastic net regression.
- `lamba`: a numeric value defining the amount of shrinkage. Should be specify by analyst.

In penalized regression, you need to specify a constant `lambda` to adjust the amount of the coefficient shrinkage. The best `lambda` for your data, can be defined as the `lambda` that minimize the cross-validation prediction error rate. This can be determined automatically using the function `cv.glmnet()`.

In the following sections, we start by computing ridge, lasso and elastic net regression models. Next, we'll compare the different models in order to choose the best one for our data.

The best model is defined as the model that has the lowest prediction error, RMSE (Chapter @ref(regression-model-accuracy-metrics)).

Computing ridge regression

```
# Find the best lambda using cross-validation
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 0)
# Display the best lambda value
cv$lambda.min
```

```
## [1] 0.758
```

```
# Fit the final model on the training data
model <- glmnet(x, y, alpha = 0, lambda = cv$lambda.min)
# Display regression coefficients
coef(model)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                   s0
## (Intercept)  28.69633
## crim         -0.07285
## zn            0.03417
## indus        -0.05745
## chas          2.49123
## nox         -11.09232
## rm            3.98132
## age          -0.00314
## dis          -1.19296
## rad           0.14068
## tax          -0.00610
## ptratio      -0.86400
## black         0.00937
## lstat        -0.47914
```

```
# Make predictions on the test data
x.test <- model.matrix(medv ~., test.data)[,-1]
predictions <- model %>% predict(x.test) %>% as.vector()
# Model performance metrics
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##   RMSE Rsquare
## 1 4.98   0.671
```

Note that by default, the function glmnet() standardizes variables so that their scales are comparable. However, the coefficients are always returned on the original scale.

Computing lasso regression

The only difference between the R code used for ridge regression is that, for lasso regression you need to specify the argument `alpha = 1` instead of `alpha = 0` (for ridge regression).

```
# Find the best lambda using cross-validation
set.seed(123)
cv <- cv.glmnet(x, y, alpha = 1)
# Display the best lambda value
cv$lambda.min
```

```
## [1] 0.00852
```

```
# Fit the final model on the training data
model <- glmnet(x, y, alpha = 1, lambda = cv$lambda.min)
# Dsiplay regression coefficients
coef(model)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## (Intercept)  36.90539
## crim         -0.09222
## zn            0.04842
## indus        -0.00841
## chas          2.28624
## nox         -16.79651
## rm            3.81186
## age           .
## dis          -1.59603
## rad           0.28546
## tax         -0.01240
## ptratio     -0.95041
## black        0.00965
## lstat       -0.52880
```

```
# Make predictions on the test data
x.test <- model.matrix(medv ~., test.data)[,-1]
predictions <- model %>% predict(x.test) %>% as.vector()
# Model performance metrics
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##   RMSE Rsquare
## 1 4.99   0.671
```

Computing elastic net regession

The elastic net regression can be easily computed using the `caret` workflow, which invokes the `glmnet` package.

We use `caret` to automatically select the best tuning parameters `alpha` and `lambda`. The `caret` packages tests a range of possible `alpha` and `lambda` values, then selects the best values for lambda and alpha, resulting to a final model that is an elastic net model.

Here, we'll test the combination of 10 different values for `alpha` and `lambda`. This is specified using the option `tuneLength`.

The best `alpha` and `lambda` values are those values that minimize the cross-validation error (Chapter @ref(cross-validation)).

```
# Build the model using the training set
set.seed(123)
model <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
# Best tuning parameter
model$bestTune
```

```
##   alpha lambda
## 6   0.1   0.21
```

```
# Coefficient of the final model. You need
# to specify the best lambda
coef(model$finalModel, model$bestTune$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  33.04083
## crim         -0.07898
## zn            0.04136
## indus        -0.03093
## chas          2.34443
## nox         -14.30442
## rm            3.90863
## age           .
## dis          -1.41783
## rad           0.20564
## tax          -0.00879
## ptratio      -0.91214
## black         0.00946
## lstat        -0.51770
```

```
# Make predictions on the test data
x.test <- model.matrix(medv ~., test.data)[,-1]
predictions <- model %>% predict(x.test)
# Model performance metrics
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

```
##   RMSE Rsquare
## 1 4.98   0.672
```

Comparing the different models

The different models performance metrics are comparable. Using lasso or elastic net regression set the coefficient of the predictor variable age to zero, leading to a simpler model compared to the ridge regression, which include all predictor variables.

✓ All things equal, we should go for the simpler model. In our example, we can choose the lasso or the elastic net regression models.

Note that, we can easily compute and compare ridge, lasso and elastic net regression using the caret workflow.

caret will automatically choose the best tuning parameter values, compute the final model and evaluate the model performance using cross-validation techniques.

Using caret package

0. **Setup a grid range of lambda values**:

```
lambda <- 10^seq(-3, 3, length = 100)
```

1. **Compute ridge regression**:

```
# Build the model
set.seed(123)
ridge <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 0, lambda = lambda)
  )
# Model coefficients
coef(ridge$finalModel, ridge$bestTune$lambda)
# Make predictions
predictions <- ridge %>% predict(test.data)
# Model prediction performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

2. **Compute lasso regression**:

```
# Build the model
set.seed(123)
lasso <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneGrid = expand.grid(alpha = 1, lambda = lambda)
  )
# Model coefficients
coef(lasso$finalModel, lasso$bestTune$lambda)
# Make predictions
predictions <- lasso %>% predict(test.data)
# Model prediction performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

3. **Elastic net regression**:

```
# Build the model
set.seed(123)
elastic <- train(
  medv ~., data = train.data, method = "glmnet",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
  )
```

```
# Model coefficients
coef(elastic$finalModel, elastic$bestTune$lambda)
# Make predictions
predictions <- elastic %>% predict(test.data)
# Model prediction performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  Rsquare = R2(predictions, test.data$medv)
)
```

4. **Comparing models performance**:

The performance of the different models - ridge, lasso and elastic net - can be easily compared using `caret`. The best model is defined as the one that minimizes the prediction error.

```
models <- list(ridge = ridge, lasso = lasso, elastic = elastic)
resamples(models) %>% summary( metric = "RMSE")
```

```
##
## Call:
## summary.resamples(object = ., metric = "RMSE")
##
## Models: ridge, lasso, elastic
## Number of resamples: 10
##
## RMSE
##         Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
## ridge   3.10    3.96   4.38 4.73    5.52 7.43    0
## lasso   3.16    4.03   4.39 4.73    5.51 7.27    0
## elastic 3.13    4.00   4.37 4.72    5.52 7.32    0
```

✓ It can be seen that the elastic net model has the lowest median RMSE.

Discussion

In this chapter we described the most commonly used penalized regression methods, including ridge regression, lasso regression and elastic net regression. These methods are very useful in a situation, where you have a large multivariate data sets.

References

Bruce, Peter, and Andrew Bruce. 2017. *Practical Statistics for Data Scientists*. O'Reilly Media.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2014. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

☆ ☆ ☆ ☆ ☆ 0 Note

Recommended for You!

The fields marked with a * are required !

Add a comment

Name

Visitor

Message

*

Preview

* Code de vérification
What is the result of 5 + ten?

Submit Reset

Sign in

Login

Login

Password

Password

Auto connect

✓

Sign in

Register

Forgotten password

Welcome!

Want to Learn More on R Programming and Data Science?

Follow us by Email

Subscribe

by FeedBurner

on Social Networks

Ads by Google

Data Analysis R

Data Set Example

Sthda Lasso

Project Analysis

factoextra

survminer

ggpubr

ggcorrplot

fastqcr

Our Books

Guest Book

I've been using R to perform survival analysis for several years now and discovered the survminer package a couple of days ago via the blog "R-addict". It is by far the best package around for produci... [Read more]

By *Visitor*

Guest Book

R-Bloggers

Newsletter Email

Recommended for you

ggplot2.customize : How to personalize ea...

www.sthda.com

survminer R package: Survival Data Analysis...

www.sthda.com

Running RStudio and Setting Up Your Worki...

www.sthda.com

AddThis