Regularization in Regression

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Co-Direcor, Center for Biomedical Informatics and Health AI
Rutgers University – New Jersey Medical School

2025-07-30

Bias-Variance tradeoff

In statistics and machine learning, the **bias–variance tradeoff** is the property of a model that the variance of the parameter estimated across samples can be reduced by increasing the bias in the estimated parameters.

The **bias–variance dilemma** or **bias–variance problem** is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set

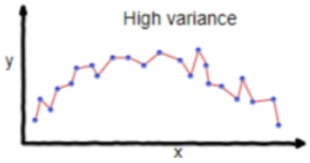
(Source: Wikipedia)

Bias-Variance tradeoff

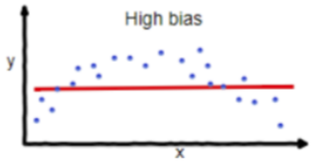
The **bias** is an error from faulty assumptions or mispecification of the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).

The **variance** is an error from sensitivity to small fluctuations in the training set. High variance may result from an algorithm modeling the random noise in the training data (overfitting).

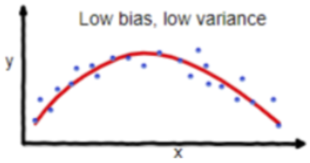
Bias-Variance tradeoff



overfitting

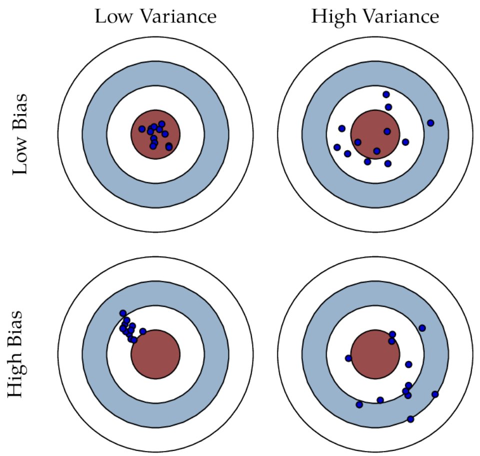


underfitting



Good balance

Bias-Variance tradeoff



Bias-Variance tradeoff

The bias–variance tradeoff is a central problem in supervised learning. Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data.

(Source: Wikipedia)

Bias-Variance tradeoff

Unfortunately, it is typically impossible to do both simultaneously. High-variance learning methods may be able to represent their training set well but are at risk of overfitting to noisy or unrepresentative training data. In contrast, algorithms with high bias typically produce simpler models that may fail to capture important regularities (i.e. underfit) in the data.

(Source: Wikipedia)

Regularization in Machine Learning

In regression analysis, the features are estimated using coefficients while modeling. In small sample sizes or noisy data coefficient estimates could be anecdotally incorrect (e.g., overfitting) or innacurate.

If the estimates can be restricted, penalized, or shrunk towards zero, then the impact of insignificant features might be reduced and would prevent models from high variance with a stable fit.[1]

[1] Adapted from: https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning

Regularization in Machine Learning

Regularization is the most used technique to penalize complex models in machine learning, it is deployed for reducing overfitting (or, contracting generalization errors) by putting small network weights into the model (adding a small amount of biad). Also, it enhances the performance of models for new inputs.[2]

[2]Source: https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning

Regularization in Machine Learning

Examples of regularization in machine learning, include:

- Regression: Penalizing coefficients to create parsimonious models (variable selection)
- K-means: Restricting the segments for avoiding redundant groups.
- Neural networks: Confining the complexity (weights) of a model.
- Random forests: Reducing the depths of tree and branches (new features)
- Neural networks: Confining the complexity (weights) of a

Ridge regression

Ridge regression **regularizes** (shrinks) coefficients by imposing a penalty on their size. The ridge coefficients minimize a penalized sum of squared error:

$$\hat{\beta}^{ridge} = \inf_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\},$$

where $\lambda \geq 0$ is a parameter that controls the shrinkage. The larger the value of λ the more shrinkage (towards 0).

Ridge regression

Or in matrix form, ridge regression minimizes:

$$\hat{\beta}^{ridge} = \inf_{\beta} \left\{ (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) + \lambda \beta^T \beta \right\}.$$

With a little work, the ridge regression solution can be shown to be:

$$\hat{\beta}^{ridge} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_N)^{-1} \mathbf{X}^T \mathbf{y}$$

Ridge regression

The regularization for ridge regression, $\lambda\beta^T\beta$, is usually denoted as an **L2 regularization** or **L2 penalty**, as it adds a penalty which is equal to the square of the magnitude of coefficients. Support Vector Machines (SVMs) also implement this method.

L2 regularization can deal with multicollinearity problems (independent variables are highly correlated) through constricting the coefficient while keeping all the variables in a model.

However, L2 regularization is not an effective method for selecting relevant predictors (or removing redundant parameters). We will later use a **L1 regularization** for this purpose.

Ridge regression: a Bayesian perspective

Ridge regression also has a clear Bayesian interpretation. It can be shownthat the Ridge penalty can be interpreted as a ‘zero’ prior (Normal prior with zero mean), and the λ is related to the variance of the prior.

Lasso regression

Lasso (Least Absolute Shrinkage and Selection Operator) regression also **regularizes** coefficients by imposing a penalty on their size, but it uses an **L1** penalty. The lasso coefficients minimize the following cost function:

$$\hat{\beta}^{lasso} = \inf_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right\},$$

where $\alpha \geq 0$ is a parameter that controls the shrinkage. The larger the value of α the more shrinkage (towards 0).

Lasso regression

Notice the similarity to the ridge regression problem: the L2 ridge penalty $\sum_{j=1}^{p} \beta_j^2$ is replaced by the L1 lasso penalty $\sum_{j=1}^{p} |\beta_j|$.

This latter constraint makes the solutions nonlinear in the y_i, and there is no closed form expression for the lasso as was the case in ridge regression.

Because of the nature of the constraint, making α sufficiently small will cause some of the coefficients to be exactly zero. Thus the lasso does a kind of continuous subset selection, or conducts a **variable selection**.

Lasso vs Ridge regression

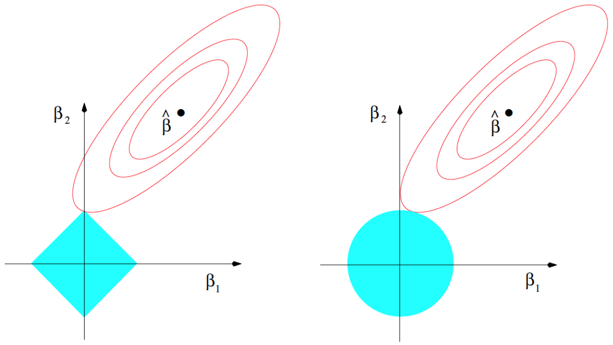
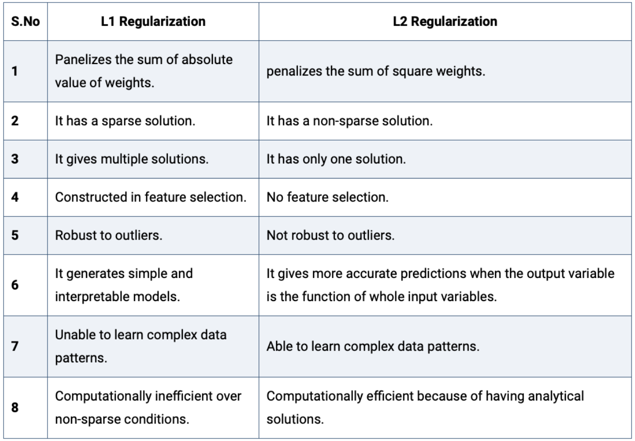


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \le t$ *and* $\beta_1^2 + \beta_2^2 \le t^2$*, respectively, while the red ellipses are the contours of the least squares error function.*

(Elements of Statistical Learning, Hastie, Tibshirani, Friedman, by Springer)

Lasso vs Ridge regression



S.No	L1 Regularization	L2 Regularization
1	Panelizes the sum of absolute value of weights.	penalizes the sum of square weights.
2	It has a sparse solution.	It has a non-sparse solution.
3	It gives multiple solutions.	It has only one solution.
4	Constructed in feature selection.	No feature selection.
5	Robust to outliers.	Not robust to outliers.
6	It generates simple and interpretable models.	It gives more accurate predictions when the output variable is the function of whole input variables.
7	Unable to learn complex data patterns.	Able to learn complex data patterns.
8	Computationally inefficient over non-sparse conditions.	Computationally efficient because of having analytical solutions.

(https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning)

Elastic net regularization

Which should I choose? Ridge or Lasso? Well, why do I have to choose! Instead use the **Elastic Net** that minimizes:

$$\hat{\beta}^{elastic\ \ net} = \inf_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \alpha \sum_{j=1}^{p} |\beta_j| + \lambda \sum_{j=1}^{p} \beta_j^2 \right\},$$

for some $\alpha \geq 0$ and $\lambda \geq 0$.

The quadratic penalty term makes the loss function strongly convex, and it therefore has a unique minimum. The elastic net method includes OLS, Lasso, and Ridge regression by setting either $\alpha = 0$, $\lambda = 0$, or both to 0.

Regression Regularization in R: glmnet

We can use the **glmnet** package to apply regularization in R:

```r
install.packages("glmnet")
```

Regression Regularization in R: glmnet

The default model used in the package is the "least squares" regression model and glmnet actually optimizes:

$$\hat{\beta}^{elastic\ \ net} = \inf_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \left(\alpha \sum_{j=1}^{p} |\beta_j| + (1-\alpha) \sum_{j=1}^{p} \beta_j^2 \right) \right\},$$

with $\alpha = 1$ as a default (so Lasso!).

Regression Regularization in R: glmnet

Using the quick start example from the package:

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-10
```

```r
data(QuickStartExample)
x <- QuickStartExample$x
y <- QuickStartExample$y
```

Regression Regularization in R: glmnet

We fit the model using the most basic call to glmnet.

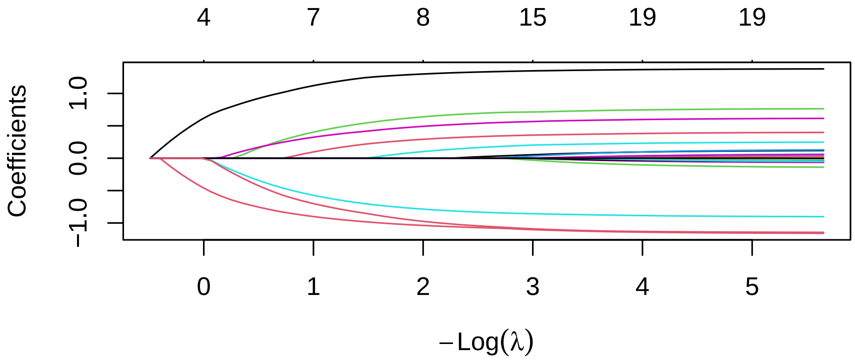
```r
fit <- glmnet(x, y)
```

fit is an object of class glmnet that contains all the relevant information of the fitted model for further use. We do not encourage users to extract the components directly. Instead, various methods are provided for the object such as plot, print, coef and predict that enable us to execute those tasks more elegantly.

Regression Regularization in R: glmnet

We can visualize the coefficients by executing the plot method:

```r
plot(fit)
```



Regression Regularization in R: glmnet

A summary of the glmnet path at each step is displayed if we just enter the object name or use the print function:

```r
print(fit)
```

```
##
## Call:  glmnet(x = x, y = y)
##
##    Df  %Dev  Lambda
## 1   0  0.00 1.63100
## 2   2  5.53 1.48600
## 3   2 14.59 1.35400
## 4   2 22.11 1.23400
## 5   2 28.36 1.12400
## 6   2 33.54 1.02400
## 7   4 39.04 0.93320
```

Regression Regularization in R: glmnet

We can obtain the model coefficients at one or more *lambda*'s within the range of the sequence:

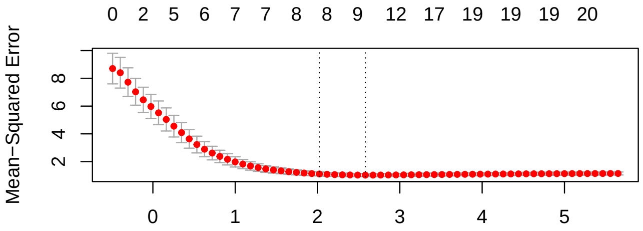
```r
coef(fit, s = 0.1)
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##                        s=0.1
## (Intercept)  0.150928072
## V1           1.320597195
## V2           .
## V3           0.675110234
## V4           .
## V5          -0.817411518
## V6           0.521436671
```

Regression Regularization in R: glmnet

The function **glmnet** returns a sequence of models for the users to choose from. **Cross-validation** is perhaps the simplest and most widely used method to select a model. **cv.glmnet** is the main function to do cross-validation here, along with various supporting methods such as plotting and prediction.

```r
cvfit <- cv.glmnet(x, y)
plot(cvfit)
```



Regression Regularization in R: glmnet

We can get the value of λ_{min} and the model coefficients:

```
cvfit$lambda.min
```

```
## [1] 0.07569327
```

```
coef(cvfit, s = "lambda.min")
```

```
## 21 x 1 sparse Matrix of class "dgCMatrix"
##               lambda.min
## (Intercept)   0.14867414
## V1            1.33377821
## V2            .
## V3            0.69787701
## V4            .
## V5           -0.83726751
## V6            0.54334327
## V7            0.02668633
## V8            0.33741131
## V9            .
## V10           .
## V11           0.17105029
## V12           .
## V13           .
## V14          -1.07552680
## V15
```

Example: Logistic Regression Elastic Net

Logistic regression is a widely-used model when the response is binary. Suppose the response variable y takes values $\{0,1\}$. We model

$$P(y = 1|\mathbf{X}) = \frac{e^{\mathbf{X}\beta}}{1 + e^{X\beta}},$$

which can be written in the following form:

$$\log \frac{P(y = 1|\mathbf{X})}{P(y = 0|\mathbf{X})} = \mathbf{X}\beta,$$

the so-called "logistic" or log-odds transformation.

Example: Logistic Regression Elastic Net

We seek to minimize the following loss function:

$$\inf_{\beta} \left\{ -\frac{1}{N} \sum_{i=1}^{N} y_i \log(\mathbf{X}\beta) - \log\left(1 + e^{\mathbf{X}\beta}\right) + \lambda \left(\alpha \sum_{j=1}^{p} |\beta_j| + (1-\alpha) \sum_{j=1}^{p} \beta_j^2 \right) \right\}$$

where the right hand side is the loss function for standard logistic regression, and the right hand side is the elastic net regularization.

Logistic regression is often plagued with degeneracies when $p > N$ and exhibits wild behavior even when N is close to p; the elastic net penalty alleviates these issues by regularizing and selecting variables.

Center for Data Science

Example: Logistic Regression Elastic Net

Using the example dataset from the glmnet package:

```r
library(glmnet)
data(BinomialExample)
x <- BinomialExample$x
y <- BinomialExample$y
```

Example: Logistic Regression Elastic Net

Set family option to “binomial” in the glmnet function:

```
fit <- glmnet(x, y, family = "binomial")
```

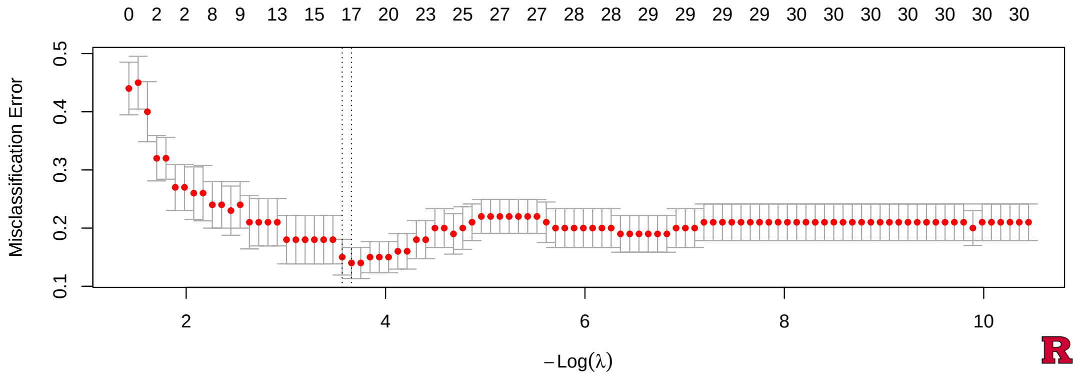
The code below uses misclassification error as the criterion for 10-fold cross-validation:

```
cvfit <- cv.glmnet(x, y, family = "binomial",
                         type.measure = "class")
```

Example: Logistic Regression Elastic Net

Now we can plot the cross-validation results and find the 'best' λ_{min}:

```r
plot(cvfit)
```



Example: Logistic Regression Elastic Net

```r
coef(cvfit, s = "lambda.min")
```

```
## 31 x 1 sparse Matrix of class "dgCMatrix"
##              lambda.min
## (Intercept)  0.23359471
## V1           .
## V2           0.46478565
## V3          -0.38040567
## V4          -0.91832966
## V5          -0.11277044
## V6          -0.69960174
## V7           .
## V8          -0.38827100
## V9           0.52283340
## V10         -1.08167257
## V11         -0.01394248
## V12          .
## V13          .
## V14          .
## V15          .
## V16          0.15154656
## V17          .
## V18          .
## V19          .
## V20          .
## V21          .
```

Session Info

```r
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.5
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3.12.1
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] glmnet_4.1-10 Matrix_1.7-3
##
## loaded via a namespace (and not attached):
##  [1] digest_0.6.37     codetools_0.2-20  fastmap_1.2.0     shape_1.4.6.1
##  [5] xfun_0.52         lattice_0.22-7    splines_4.5.1     iterators_1.0.14
##  [9] knitr_1.50        htmltools_0.5.8.1 rmarkdown_2.29    tinytex_0.57
## [13] cli_3.6.5         foreach_1.5.2     grid_4.5.1        compiler_4.5.1
```