Penalized Regression

EN5422/EV4238 | Fall 2023

w04_penalized.pdf

(Week 4 – 1/1)

Contents

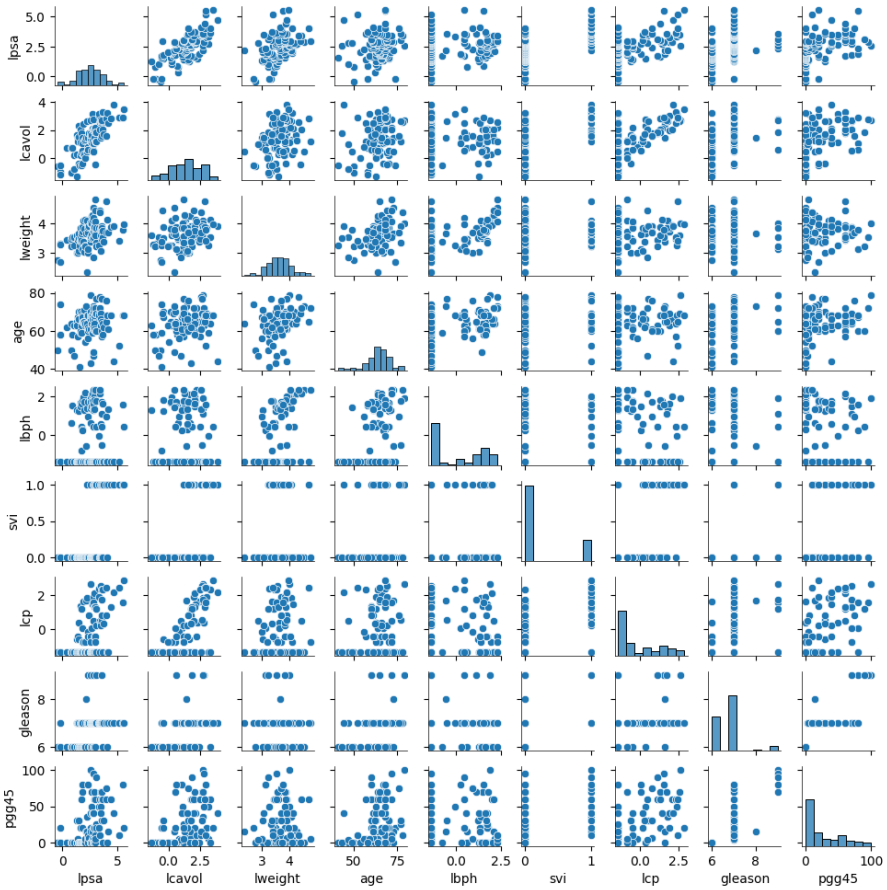
1 Shrinkage and Penalized Regression Intro

1.1 Prostate Cancer Data

The Elements of Statistical Learning (ESL) text has a description of a prostate cancer dataset used in a study by Stamey et al. (1989). They examined the correlation between the level of prostate-specific antigen and a number of clinical measures in mean who were about to receive a radical prostatectomy.

The variables are:

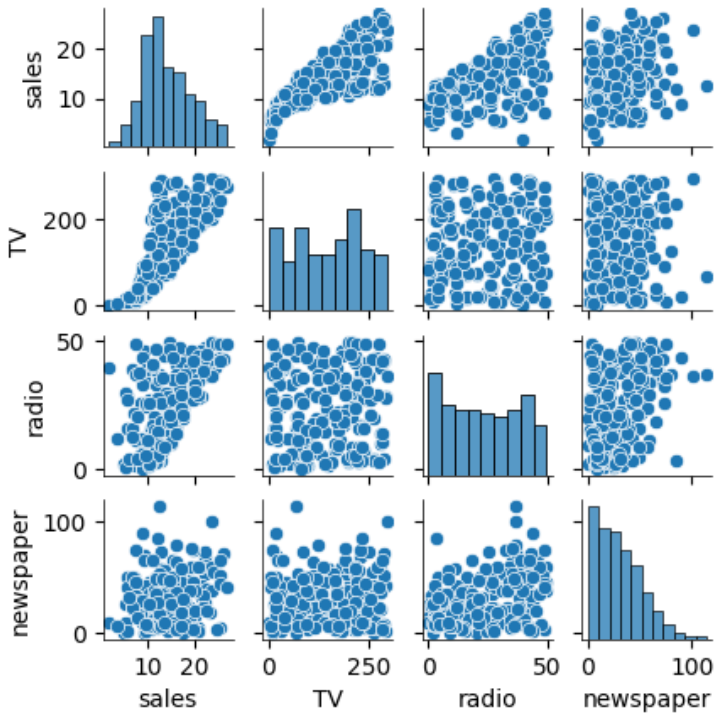
- log cancer volume (lcavol)
- log prostate weight (lweight)
- age
- log of the amount of benign prostatic hyperplasia (lbph)
- seminal vesicle invasion (svi)
- log of capsular penetration (lcp)
- gleason score (gleason)
- Percent of Gleason scores 4 or 5 (pgg45)
- outcome variable is the log of prostate-specific antigen (lpsa)



1.2 Advertising Data

The Introduction to Statistical Learning (ISL) text has some data on advertising.
These data give the sales of a product (in thousands of units) under advertising budgets (in thousands of dollars) of TV, radio, and newspaper.

The goal is to predict sales for given TV, radio, and newspaper budget.



1.3 Linear Regression (OLS)

The standard generic form for a linear regression model is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots, + \beta_p x_p + \epsilon$$

- y is the response or dependent variable.
- $x_1, x_2, \ldots, x_p$ are called the p explanatory, independent, or predictor variables.
- The Greek letter ϵ (epsilon) is the random error variable.
- For example:

$$\text{sales} = \beta_0 + \beta_1 \times (\text{TV}) + \beta_2 \times (\text{radio}) + \beta_3 \times (\text{newspaper}) + \text{error}$$

Training data is used to estimate the *model parameters* or *coefficients*.

Producing the predictive model:

$$\hat{y}(x_1, x_2, \dots, x_p) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots, + \hat{\beta}_p x_p + \epsilon$$

- Where $\hat{\beta}_j$ are the weights assigned to each variable.
- These weights are the values that minimize the residual sum of squares (RSS) for predicting the training data.
- For example:

$$\widehat{\text{sales}} = 2.939 + 0.46 \times (\text{TV}) + 0.189 \times (\text{radio}) - 0.001 \times (\text{newpaper})$$

- The *complexity* of an OLS regression model is the *number of estimated parameters*.
  - E.g., $p + 1$ (using the notation above), where the +1 is added for the intercept.

1.4 Estimation

- The weights/coefficients (β) are the *model parameters.*
- OLS uses the weights/coefficients that minimize the RSS loss function over the training data.

$$\hat{\beta} = \arg\min_{\beta} \text{RSS}(\beta) \;\; Note: \beta \text{ is a } vector$$
$$= \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - f(\mathbf{x}_i; \beta)\right)^2$$
$$= \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_{i1} - \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right)^2$$

Note: OLS equivalently minimizes the MSE since MSE = RSS/n.

1.4.1 Matrix notation

$$f(\mathbf{x}; \beta) = \mathbf{x}^{\mathrm{T}} \beta$$

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \quad X = \begin{bmatrix} 1 & X_{11} & X_{12} & X_{13} & \cdots & X_{1p} \\ 2 & X_{21} & X_{22} & X_{23} & \cdots & X_{2p} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & X_{n3} & \cdots & X_{np} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

$$\text{RSS}(\beta) = (\text{Y} - \text{X}\beta)^{\mathrm{T}}(\text{Y} - \text{X}\beta)$$

$$\frac{\partial \text{RSS}(\beta)}{\partial \beta} = 2\text{X}^{\mathrm{T}}(\text{Y} - \text{X}\beta) \rightarrow \text{X}^{\mathrm{T}}\text{Y} = \text{X}^{\mathrm{T}}\text{X}\beta \rightarrow \hat{\beta} = (\text{X}^{\mathrm{T}}\text{X})^{-1}\text{X}^{\mathrm{T}}\text{Y}$$

1.5 Some Problems with least squares estimates

There are a few problems with using OLS to estimate the regression parameters (coefficients):

- *Prediction Accuracy*
  - The OLS in high dimensional data may have low bias but can suffer from large variance.
  - Prediction accuracy can sometimes be improved by shrinking or setting some coefficients to zero.
  - By doing so we sacrifice a little bit of bias to reduce the variance of the predicted values, and hence may improve the overall prediction accuracy.
  - Some predictors may not have any predictive value and only increase noise.
- *Interpretation*
  - With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects. In order to get the "big picture", we are willing to sacrifice some of the small details
    - When $p > n$, OLS won't work at all

1.6 Improving Least Squares

We will examine 3 standard approaches to improve on OLS estimates.

1. Subset Selection
   a. Only use a subset of predictors, but estimate with OLS.
   b. Example: *best subsets, forward step-wise*.
2. Shrinkage/Penalize/Regularized Regression
   a. Instead of an "all or nothing" approach, shrinkage methods force the coefficients closer toward 0.
   b. Example: ridge, lasso, elastic net.
3. Dimension Reduction with Derived Inputs
   a. Use a subset of linearly transformed predictors.
   b. Examples: PCA, PLS

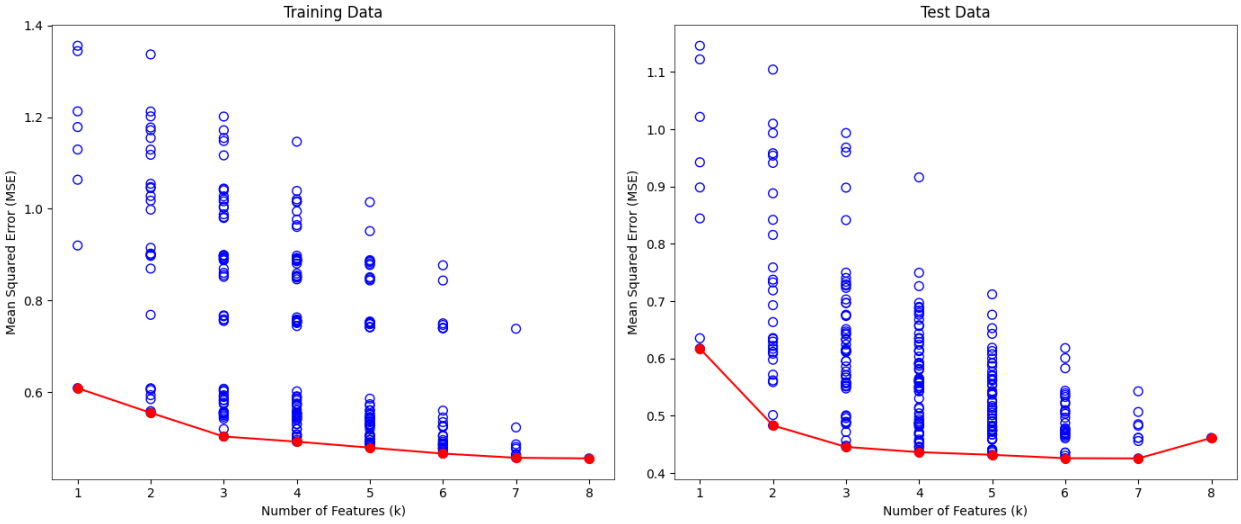
All three methods introduce some additional bias in order to reduce variance and *hopefully* improve prediction.

2 Subset Selection

Subset selection methods attempt to find the best subset of predictors to use in the model.

- Best Subsets finds the best (usually in terms of minimum RSS/MSE) combination of k predictors – k is the *tuning parameter.*

- Stepwise Selectors takes a greedy approach by sequentially adding (forward) or deleting (backward) the predictor that most improves the fit.
    - This is a computational necessity for high dimensional data.
    - Tuning parameter options:
        - Number of predictor (k)
        - Inclusion/Exclusion criteria: AIC/BIC, adjusted R^2, p-values, etc.



Note: Subset selection methods remove predictors by setting their coefficients to 0 (e.g., $\hat{\beta} = 0$)

- These "all or nothing" approaches can be very unstable. A small change in the data can completely change the model.

predictor	lm	best_subset	bootstrap
(Intercept)	0.43	-1.05	-0.33
lcavol	0.58	0.63	0.51
lweight	0.61	0.74	0.54
age	-0.02	0.00	0.00
lbph	0.14	0.00	0.14
svi	0.74	0.00	0.67
lcp	-0.21	0.00	0.00
gleason	-0.03	0.00	0.00
pgg45	0.01	0.00	0.00

3 Shrinkage Methods

Instead of an "all or nothing" approach, shrinkage methods force the coefficients closer toward 0.

- Usually this is accomplished through penalized regression where a penalty is imposed on the size of the coefficients.
- Equivalently, the size of the coefficients is *constrained* not to exceed a threshold.

3.1 Two Representations

The penalized optimization (Lagrangian form)

$$\hat{\beta} = \arg\min_{\beta}\{l(\beta) + \lambda P(\beta)\}$$

An equivalent representation is (constrained optimization)

$$\begin{aligned}\hat{\beta} &= \arg\min_{\beta} l(\beta) \text{ subject to } P(\beta) \leq t \\ &= \arg\min_{\beta: P(\beta)\leq t} l(\beta)\end{aligned}$$

where

- $l(\beta)$ is the loss function (e.g., mean squared error, negative log-likelihood).
- $P(\beta)$ is the penalty term (as a function of the model paramters).
- $\lambda \geq 0$ is the strength of the penalty.
- t is the penalty budget.

3.2 Penalties

Examples penalties:

- Ridge Penalty

$$P(\beta) = \sum_{j=1}^{p} |\beta_j|^2 = \beta^{\mathrm{T}}\beta$$

- Lasso Penalty

$$P(\beta) = \sum_{j=1}^{p} |\beta_j|$$

- Best Subsets

$$P(\beta) = \sum_{j=1}^{p} |\beta_j|^0 = \sum_{j=1}^{p} 1_{(\beta_j \neq 0)}$$

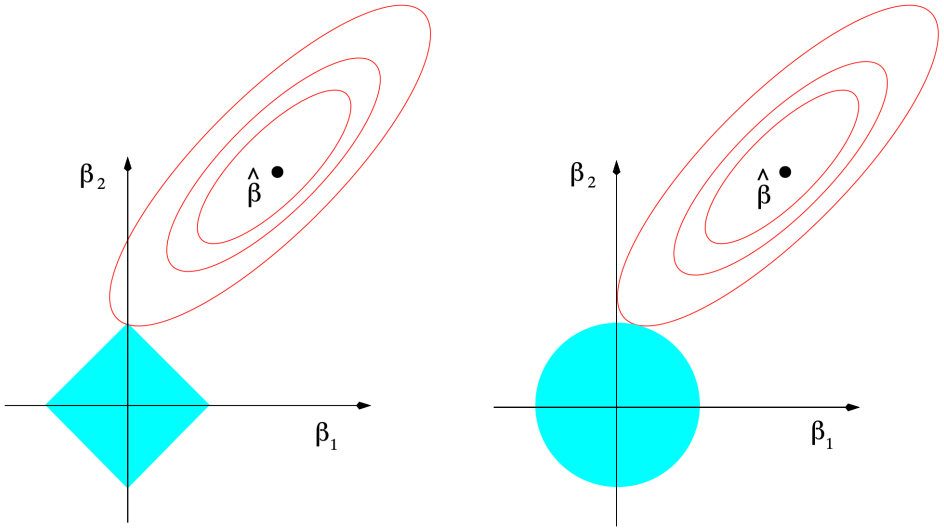
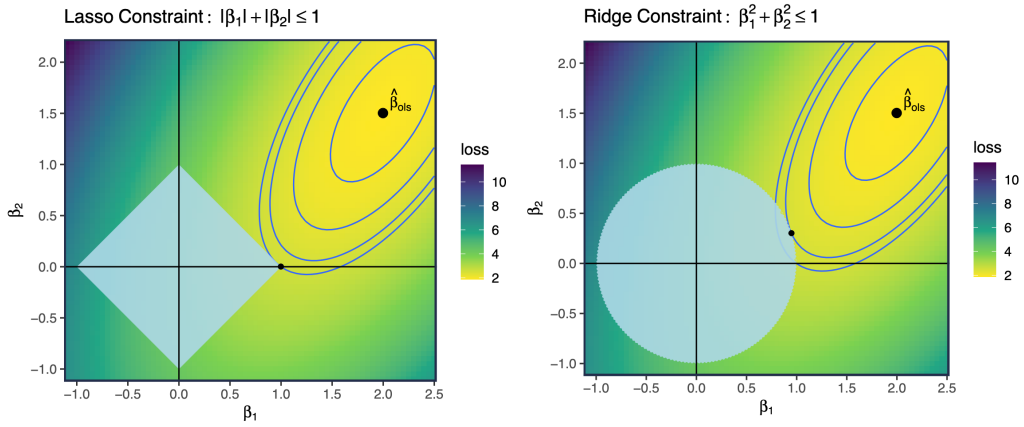


FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|\beta_1| + |\beta_2| \leq t$ *and* $\beta_1^2 + \beta_2^2 \leq t^2$*, respectively, while the red ellipses are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



4 Ridge Regression

For ridge regression

$$l(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 = \text{MSE}$$

$$P(\beta) = \sum_{j=1}^{p}|\beta_j|^2 \quad \text{(Notice that the intercept, } \beta_0\text{, is not penalized)}$$

Note: Watch for how software defines the loss. Some use MSE, others use SSE = n*MSE. E.g., if loss is RSS = SSE = n*MSE, then $P(\beta)$ is a function of the sample size n!

So the ridge solution becomes:

$$\begin{aligned}\hat{\beta}_\lambda^{ridge} &= \arg\min_\beta \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|^2 \\ &= \arg\min_\beta \ \text{MSE}(\beta) + \lambda\sum_{j=1}^{p}|\beta_j|^2 \\ &= \arg\min_\beta \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + n\lambda\sum_{j=1}^{p}|\beta_j|^2 \\ &= \arg\min_\beta \ \text{RSS}(\beta) + n\lambda\sum_{j=1}^{p}|\beta_j|^2\end{aligned}$$

Your Turn #1

1. What happens when $\lambda = 0$?
2. What happens when $\lambda \uparrow \infty$?
3. Why is it important to scale the predictor variables?

4.1 Scaling

Because the penalty is based on the magnitude of the coefficients, it is important to *scale* the predictors so they are all treated equally.

- This is because predictors on different scales will be penalized with different strengths.
- The type of scaling that allows equal treatment is to **divide each predictor by its standard deviation**.
    - The resulting predictors should have the property: $x_j^T x_j = c, \forall j$.

Consider the *advertising data*:

```
# Load the dataset
url = "https://raw.githubusercontent.com/hardikkamboj/An-Introduction-to-Statistical-
Learning/d891284abad8c95f84d9a62dc352f261ca3375b1/data/Advertising.csv"
df_ad = pd.read_csv(url)

# Prepare the data for regression
X = df_ad[['TV', 'radio', 'newspaper']]
X = sm.add_constant(X)  # Add an intercept (constant) to our model
y = df_ad['sales']

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Extract the table similar to broom's tidy()
summary_table = pd.DataFrame({
    'term': model.params.index,
    'estimate': model.params.values,
    'std.error': model.bse.values,
    'statistic': model.tvalues.values,
    'p.value': model.pvalues.values
})
```

term	estimate	std.error	statistic	p.value
const	2.938889	0.311908	9.422288	1.267295e-17
TV	0.045765	0.001395	32.808624	1.509960e-81
radio	0.188530	0.008611	21.893496	1.505339e-54
newspaper	-0.001037	0.005871	-0.176715	8.599151e-01

Note: **std.error** (standard error) of a coefficient represents the standard deviation of the sampling distribution of the coefficient, **t-statistic** in regression is a measure that helps us determine the significance of predictors, **p-value** is a measure of the evidence against a null hypothesis. In the context of linear regression, the null hypothesis for each coefficient is typically that the coefficient is equal to zero (no effect).

- While the raw magnitude of radio > TV, the t-statistic (estimate/std.error) shows that TV has a stronger effect. This is because radio has a larger standard error.
- Look at what happens if we transform newspaper spending into thousands of dollars.

```
# Scale the newspaper column
df_ad['newspaper_scaled'] = df_ad['newspaper'] / 1000

# Prepare data for regression
X = df_ad[['TV', 'radio', 'newspaper_scaled']]
X = sm.add_constant(X)  # Add an intercept (constant) to our model
y = df_ad['sales']

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Extract the table similar to broom's tidy()
summary_table = pd.DataFrame({
    'term': model.params.index,
    'estimate': model.params.values,
    'std.error': model.bse.values,
    'statistic': model.tvalues.values,
    'p.value': model.pvalues.values
})
```

term	estimate	std.error	statistic	p.value
const	2.938889	0.311908	9.422288	1.267295e-17
TV	0.045765	0.001395	32.808624	1.509960e-81
radio	0.188530	0.008611	21.893496	1.505339e-54
newspaper_scaled	-1.037493	5.871010	-0.176715	8.599151e-01

- The coefficient of newspaper now has the largest magnitude!
- However, we see that the t-statistic (and p-values) haven't changed.
    - Let's divide each predictor by its standard deviation.

```
# Scale the predictors
df_ad_scaled = df_ad.copy()
df_ad_scaled[['TV', 'radio', 'newspaper']] = df_ad[['TV', 'radio',
'newspaper']].div(standard_deviations[['TV', 'radio', 'newspaper']], axis=1)

# Prepare data for regression
X = df_ad_scaled[['TV', 'radio', 'newspaper']]
X = sm.add_constant(X)  # Add an intercept (constant) to our model
y = df_ad['sales']
```

```
# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Extract the table similar to broom's tidy()
summary_table = pd.DataFrame({
    'term': model.params.index,
    'estimate': model.params.values,
    'std.error': model.bse.values,
    'statistic': model.tvalues.values,
    'p.value': model.pvalues.values
})
```

term	estimate	std.error	statistic	p.value
const	2.938889	0.311908	9.422288	1.267295e-17
TV	3.929089	0.119758	32.808624	1.509960e-81
radio	2.799069	0.127849	21.893496	1.505339e-54
newspaper	-0.022595	0.127862	-0.176715	8.599151e-01

- Notice that the scaled coefficients are original*std.dev.

term	original	std.dev	scaled
(Intercept)	2.939	N/A	2.939
TV	0.046	85.85	3.929
Radio	0.189	14.85	2.799
Newspaper	-0.001	21.78	-0.023

- While this type of transformation on unpenalized models won't impact predictions, it will have a large effect on penalized models.

Software

- Check the description of the implementation for penalized regression models. Most will properly scale the data for you behind the scenes. But if not, then you should o so first.
- In Python, the `StandardScaler` class from the `sklearn.preprocessing` can both center and scale the predictors. Centering is perfectly fine as it only impacts the intercept term which isn't penalized.

Other Transformations

There are other types of transformations that re-scale the predictors.

1. Power transformations (e.g., Box-Cox)
2. Rank/Quantile scaling. Convert each value to its associated quantile or rank. E.g., the smallest value gets scored 1 and largest value gets scored n. Implicitly used by predictions trees.
3. Range scaling, E.g., $x' = \frac{\max - x}{\max - \min}$ to force between [0, 1].

These could all be used, b ut the statistical scaling (standard deviation) is the most common. But use whichever approach gives the best predictions!

NB: Ensure all transformations are done on the training data and applied to the test data, else data leakage will occur.

4.2 Estimation

- Ridge regression has two types of parameters that need to be estimated.
  - Model parameters: β
  - Tuning parameter: λ
- The tuning parameter λ controls the model complexity and effective degrees of freedom (edf).
- Given a specific value of λ, the model parameters β are easy to estimate as we show below.

The optimization function can be written:

$$\hat{\beta}_\lambda^{ridge} = \arg\min_\beta l(\beta) + \lambda P(\beta) = \arg\min_\beta J(\beta; \lambda)$$

where

$$J(\beta) = (\mathbf{y} - \mathbf{X}\beta)^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^{\mathrm{T}}\beta$$

And the solution must satisfy

$$\frac{\partial J(\beta)}{\partial \beta} = \frac{\partial l(\beta)}{\partial \beta} + \frac{\lambda P(\beta)}{\partial \beta} = 0$$

For Lasso regression, this becomes:

Lasso Regression Solution

$$J(\beta) = (\mathbf{y} - \mathbf{X}\beta)^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\beta) + \lambda \sum |\beta_j|$$

The challenge with Lasso is that the absolute value function isn't differentiable everywhere (specifically, it's not differentiable at 0).

The gradient for the loss (least squares term) is:

$$\frac{\partial l(\beta)}{\partial \beta} = -2\mathbf{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\beta)$$
$$\frac{\partial P(\beta)}{\partial \beta_j} = sign(\beta_j)$$

Where $sign(\beta_j)$ is:
• 1 if $\beta_j > 0$
• -1 if $\beta_j < 0$
• Any value between -1 and 1 (inclusive) if $\beta_j = 0$.

Thus, the gradient (or subgradient) condition becomes:

$$-2\mathbf{X}^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\beta) + sign(\beta_j) = 0$$

This equation doesn't have a closed-form solution like Ridge regression because of the absolute value term, and numerical optimization techniques, such as coordinate descent, are often used to solve for β in Lasso regression.

Resources: The Matrix Cookbook has some common matrix and vector derivative expression.

4.3 Ridge Regression Properties

$$\hat{\beta}_{\lambda}^{ridge} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I_p}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

- Ridge regression always works, even when $\mathbf{X}$ is not full rank because $\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I_p}$ is always invertible for $\lambda > 0$.
- Ridge tends to shrink correlated predictors together.
- For $0 < \lambda > 2\sigma^2 / \sum_j |\beta_j|^2$, ridge regression has a lower mean square prediction error than least squares (Theobald 1974)!
- (Quasi) Bayesian Interpretation: If $\beta \sim N(0, \tau^2\mathbf{I}_p)$ is the prior distribution, and τ and the standard deviation σ are assumed known, then the posterior mode (and hence mean sicne Gaussian) of β, given the data, is

$$E[\,\beta|\mathcal{D}] = \left(\mathbf{X}^{T}\mathbf{X} + \frac{\sigma^2}{\tau^2}\mathbf{I}_p\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

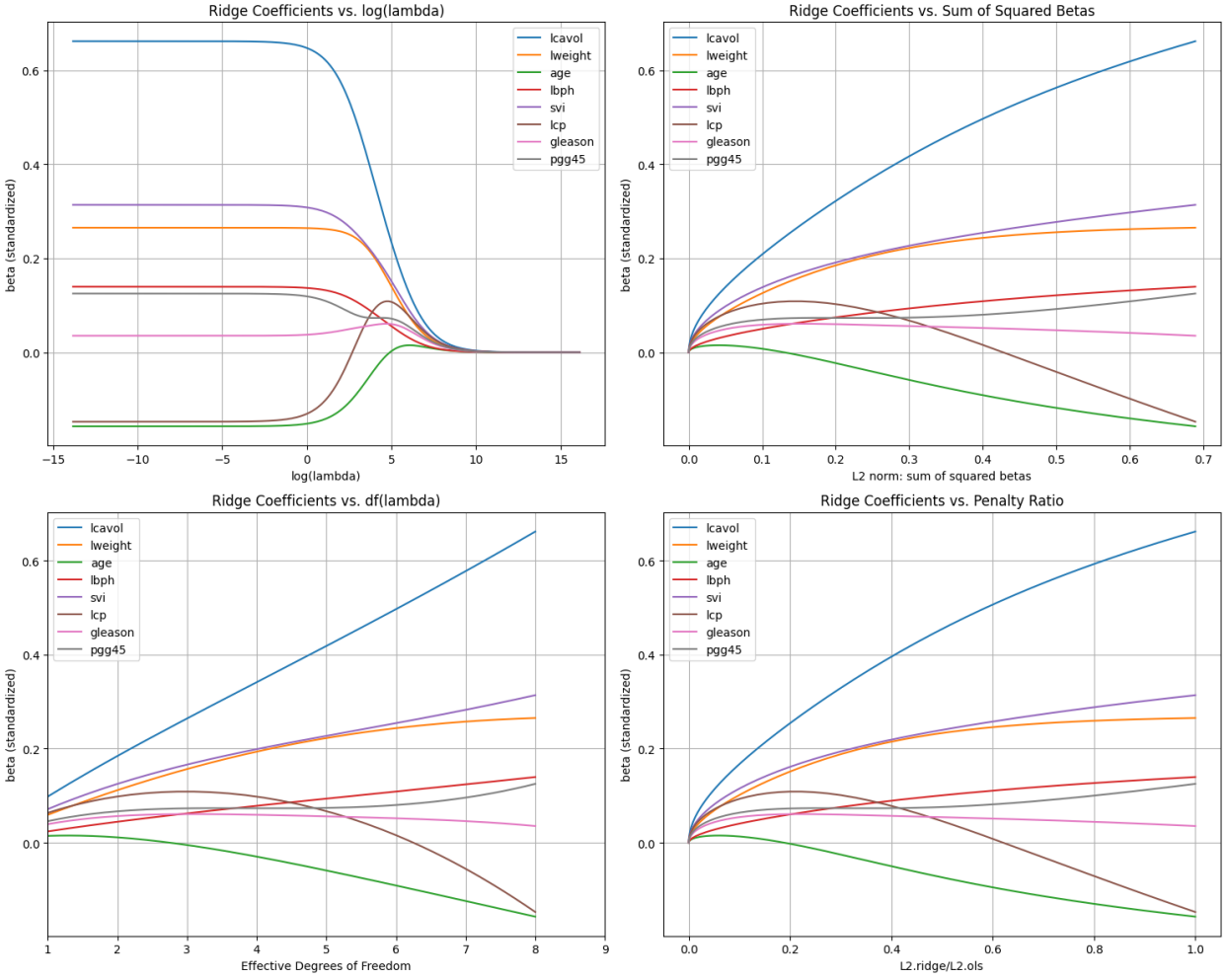
Which is equivalent to use $\lambda = \frac{\sigma^2}{\tau^2}$.

4.4 Solution Paths

Ridge Regression introduces a set of models indexed by λ.

- $\lambda = 0$ gives β^{LS}
- $\lambda = \infty$ gives $\beta_j = 0\ j = 1, \ldots, p$
- As λ goes up, variance decreases and bias increases.

It can be illustrative to plot the *coefficient path* against:



4.5 Effective Degrees of Freedom (edf)

The *tuning parameter* for a ridge regression model is the λ that controls the strength of penalty.

$$\hat{\beta}_{\lambda}^{ridge} = \arg\min_{\beta} \text{MSE}(\beta) + \lambda \sum_{j=1}^{p} |\beta_j|^2$$

The *effective degrees of freedom,* df(λ) is the trace of the hat matrix, H_λ.

EDOF Calculation for Ridge

$$\text{H}_\lambda = \mathbf{X}(\mathbf{X}^\text{T}\mathbf{X} + \lambda\text{I})^{-1}\mathbf{X}^\text{T}$$

The effective degrees of freedom is the trace of this matrix:

$$df(\lambda) = \text{trace}(\text{H}_\lambda)$$

Note: hat matrix produces the predicted values from the observed values. The trace of a matrix is the sum of the diagonal elements of that matrix.

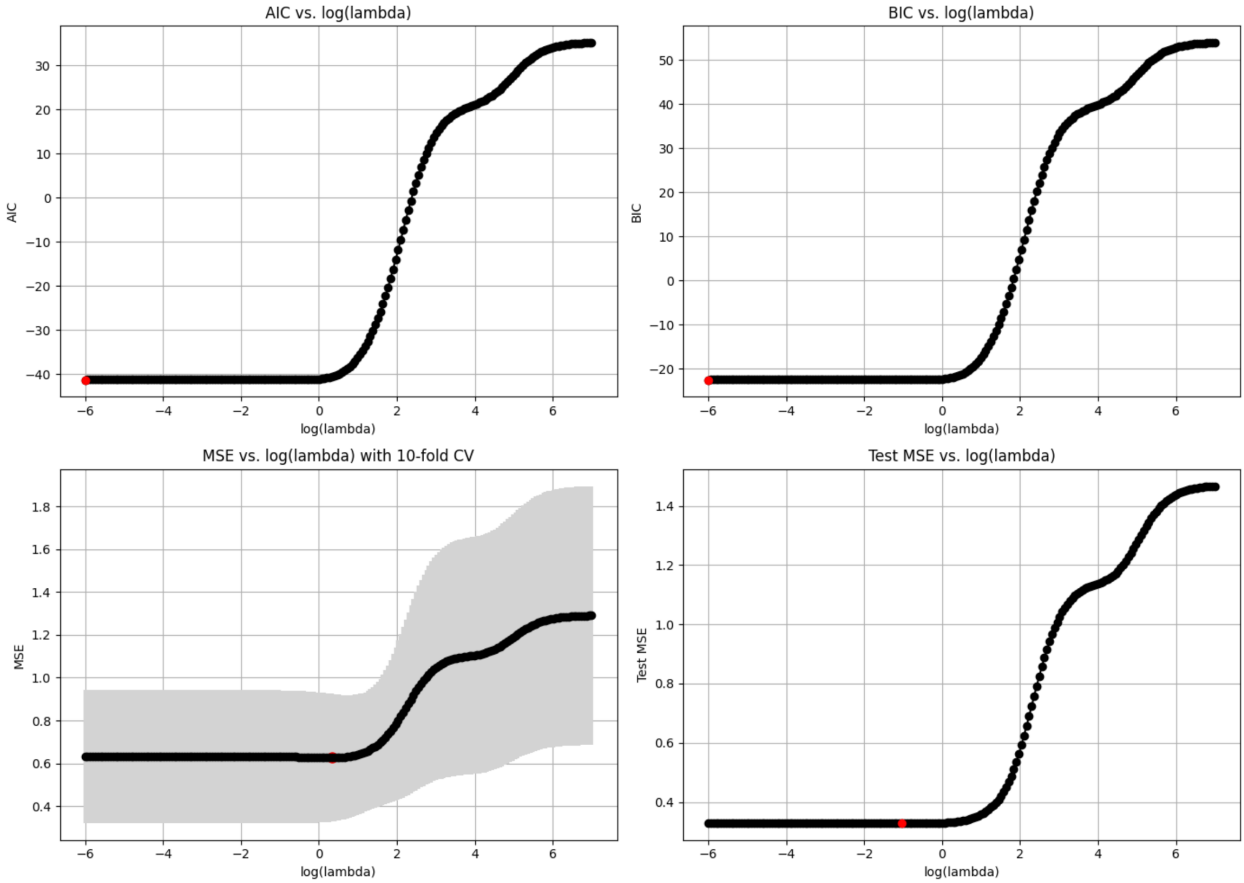
$$trace(A) = \sum_{i=1}^{n} a_{ii}$$

```
import numpy as np

# Sample data
x = np.array([1, 2])
y = np.array([3, 4])

# Design matrix with an added intercept term
X = np.column_stack([np.ones(x.shape), x])

# Compute the hat matrix
H = X @ np.linalg.inv(X.T @ X) @ X.T

# Get the trace of the hat matrix
trace_H = np.trace(H)

print("Hat matrix H:")
print(H)
print("\nTrace of Hat matrix:", trace_H)

# Regularization parameter for ridge regression
lambda_value = 1

# Compute the hat matrix for ridge regression
H_ridge = X @ np.linalg.inv(X.T @ X + lambda_value * np.eye(X.shape[1])) @ X.T

# Get the trace of the hat matrix
```

```
trace_H_ridge = np.trace(H_ridge)


print("Hat matrix H for Ridge Regression:")
print(H_ridge)
print("\nTrace of Hat matrix for Ridge Regression:", trace_H_ridge)
```

4.6 Tuning Parameter Selection

How about AIC/BIC or resampling?



4.7 Ridge Regression function in Python

There are several Python libraries that provide functions for ridge regression.

- The **sklearn** library has the function **Ridge()** in its linear_model module.

```
from sklearn.linear_model import Ridge
# alpha is equivalent to lambda in ridge regression.
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X, y)
```

- The **sklearn** Python package offers functions for elastic-net penalized regression.
- Use l1_ratio=0 for ridge regression.
- Note: input data as a matrix (or DataFrame)
- More information on using the **sklearn** package in Python for ridge, lasso, and elastic net penalized regression can be found in their documentation.

```
from sklearn.linear_model import Lasso, ElasticNet
# For Lasso
lasso_model = Lasso(alpha=1.0)
lasso_model.fit(X, y)

# For ElasticNet
# l1_ratio=1 is equivalent to Lasso; l1_ratio=0 is equivalent to Ridge.
elasticnet_model = ElasticNet(alpha=1.0, l1_ratio=0.5)
elasticnet_model.fit(X, y)
```

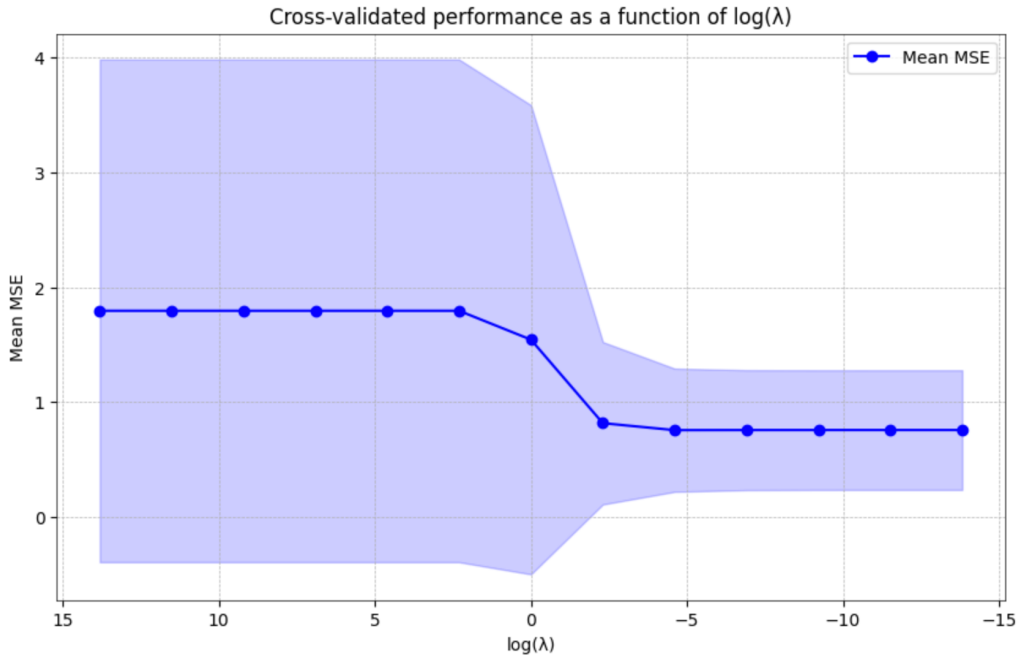
- For **sklearn**'s ridge, you would need to manually scale the features using **StandardScaler** if necessary.

4.7.1 skelarn.linear_model

Here, I demonstrate how to use **sklearn.linear_model** and CV implemented **sklearn.linear_model**.

```
from sklearn.linear_model import ElasticNetCV, ElasticNet
from sklearn.model_selection import cross_val_score
import numpy as np

# Setting seed for reproducibility
np.random.seed(2021)

# Using ElasticNetCV for 10-fold cross-validation
# l1_ratio is set to 0.5 as an example to represent ElasticNet, adjust it
as per your requirements
alphas = np.logspace(-6, 6, 13)
enet_cv = ElasticNetCV(l1_ratio=0.5, alphas=alphas, cv=10)
enet_cv.fit(X_train, y_train)

# Extracting the best alpha and its mean squared error
best_alpha = enet_cv.alpha_
mse_scores = -cross_val_score(ElasticNet(alpha=best_alpha, l1_ratio=0.5),
X_train, y_train, cv=10, scoring='neg_mean_squared_error')
mean_mse = mse_scores.mean()
```

```
std_mse = mse_scores.std()

print(f"Best alpha: {best_alpha}")
print(f"Mean MSE for best alpha: {mean_mse}")
print(f"Standard deviation of MSE for best alpha: {std_mse}")

# Since ElasticNetCV does not have the store_cv_values option like
RidgeCV,
# we will loop through the alphas and compute cross-validated MSE for each
alpha
mean_mse_scores = []
std_mse_scores = []

for alpha in alphas:
    mse_scores = -cross_val_score(ElasticNet(alpha=alpha, l1_ratio=0.5),
X_train, y_train, cv=10, scoring='neg_mean_squared_error')
    mean_mse_scores.append(mse_scores.mean())
    std_mse_scores.append(mse_scores.std())

# Convert to arrays for array operations
mean_mse_scores = np.array(mean_mse_scores)
std_mse_scores = np.array(std_mse_scores)

# Define upper and lower bounds for the point range plot
cv_lo = mean_mse_scores - std_mse_scores
cv_up = mean_mse_scores + std_mse_scores
log_alphas = np.log(alphas)

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(log_alphas, mean_mse_scores, marker='o', color='blue',
label='Mean MSE')
plt.fill_between(log_alphas, cv_lo, cv_up, color='blue', alpha=0.2)
plt.gca().invert_xaxis()  # Typically we plot largest to smallest values
of log(λ)
plt.xlabel('log(λ)')
plt.ylabel('Mean MSE')
plt.title('Cross-validated performance as a function of log(λ)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



5 Lasso

5.1 The Lasso

For lasso regression

$$l(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 = \text{MSE}$$

$$P(\beta) = \sum_{j=1}^{p}|\beta_j| \quad \text{(Notice that the intercept, } \beta_0\text{, is not penalized)}$$

So the ridge solution becomes:

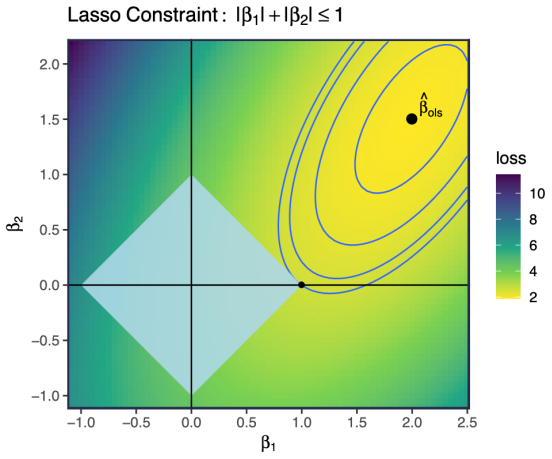
$$\hat{\beta}_\lambda^{lasso} = \arg\min_{\beta}\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

Why is it important to scale the predictor variables?

5.2 Lasso Penalty

- By using a L_1 penalty, lasso penalty can shrink some coefficients all the way to 0 (unlike the ridge penalty).

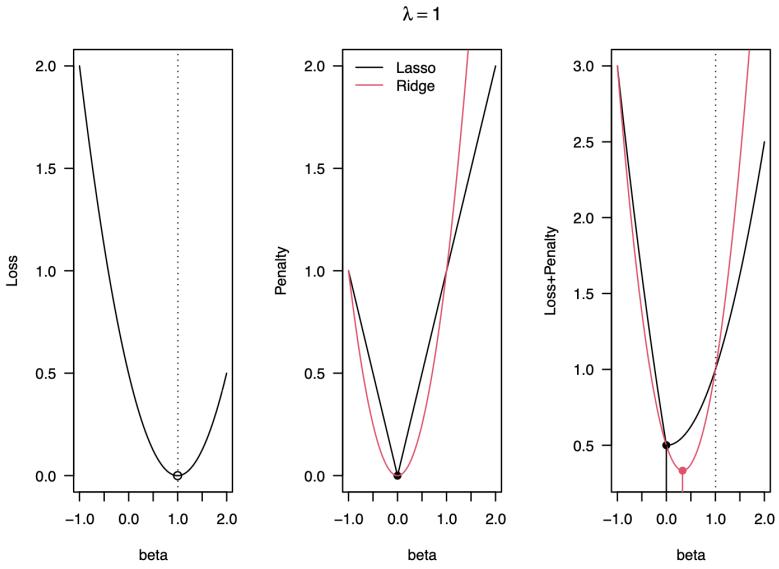
- This effectively removes predictors from the model (like the stepwise procedures), but in a type of continuous fashion.
- Lasso stands for "Least Absolute Shrinkage and Selection Operator"



5.3 Example of 1D Lasso Selection

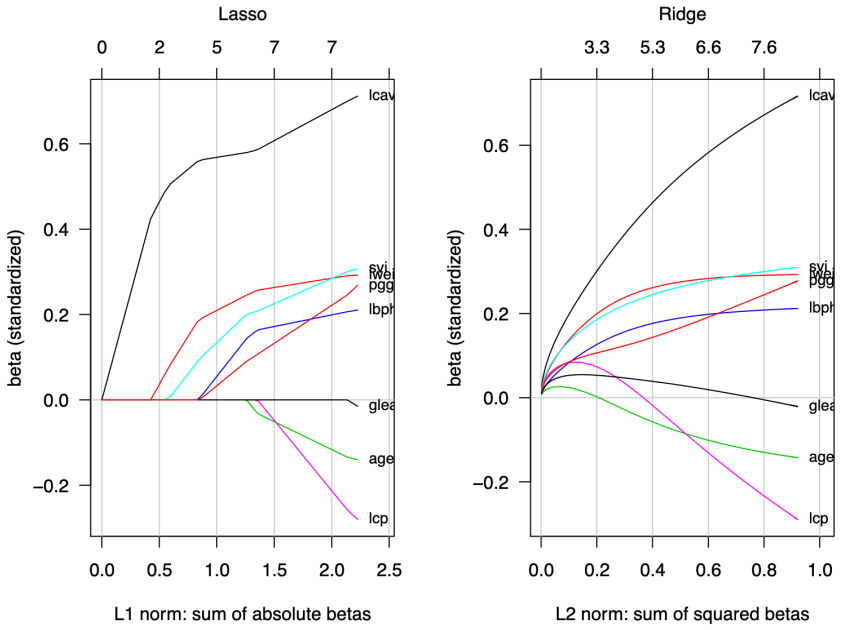
Suppose the simplified setting of fitting a loss function of $l(\beta) = \frac{1}{2}(1-\beta)^2$.

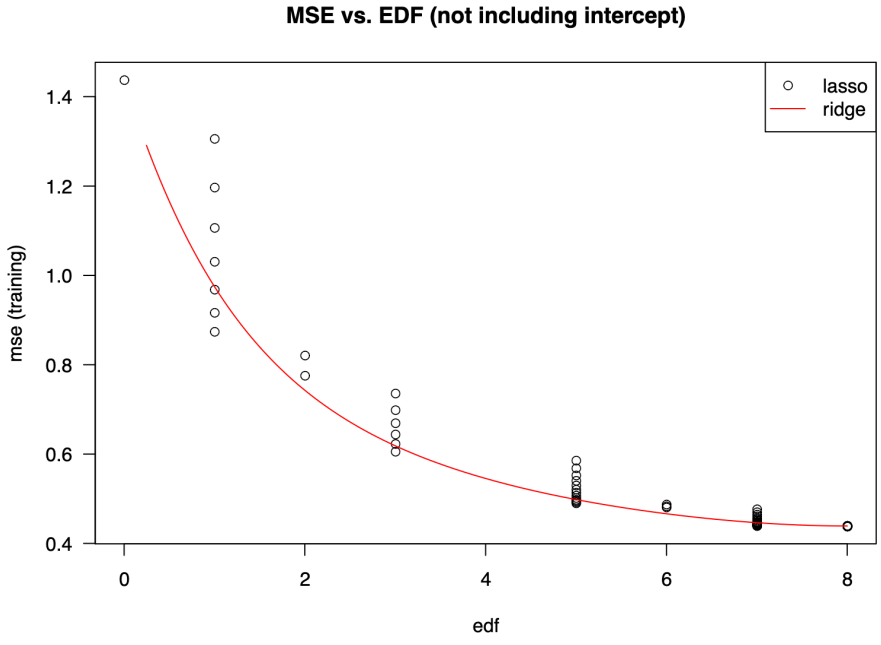
- This loss is the squared deviation from 1.
- The lasso penalty is $|\beta|$.
- The objective function is $l(\beta) + \lambda|\beta|$.



5.4 Comparing Lasso and Ridge Regression

(Prostate Cancer Data from ESL book: Figs 3.8, 3.10 and Table 3.3)



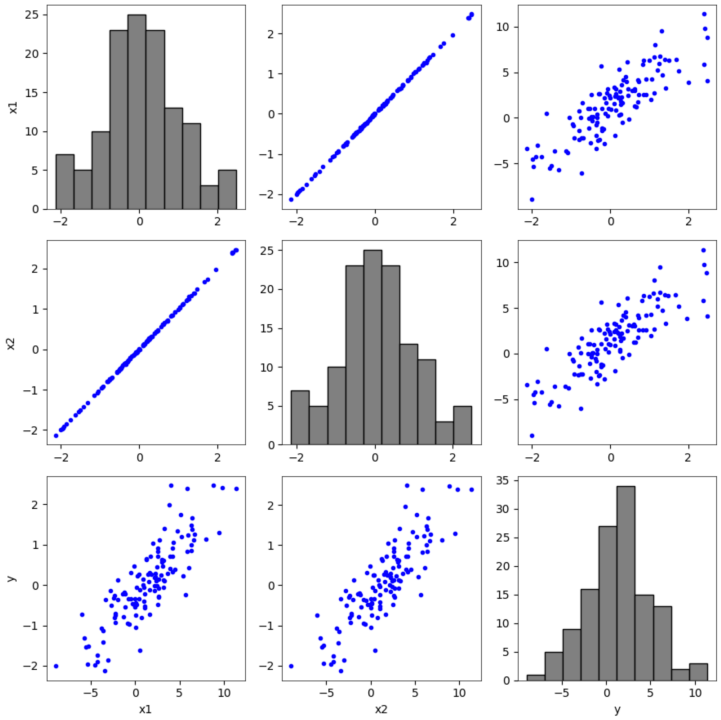


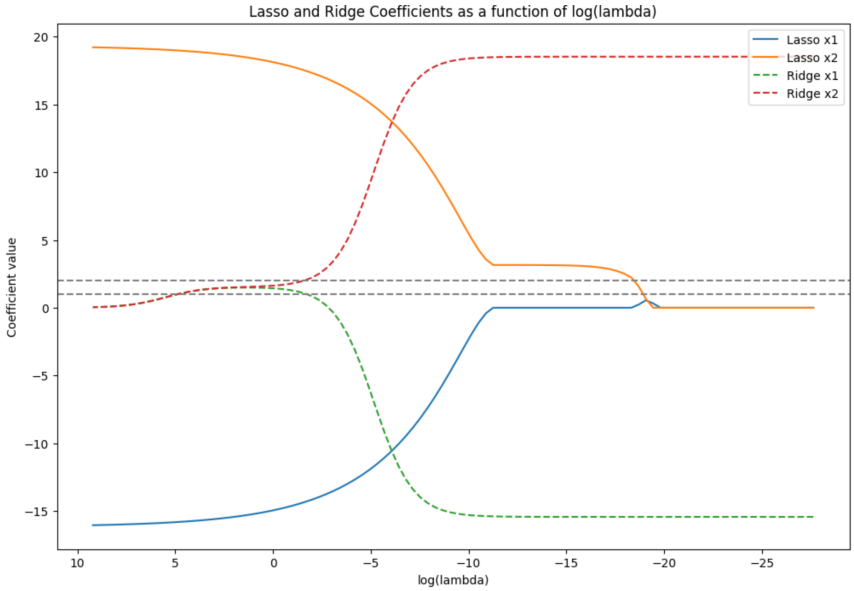
5.4.1 Example with Strong Correlation

Consider a problem with strong multicollinearity:

```
import numpy as np
import matplotlib.pyplot as plt

# Seed for reproducibility
```

```
np.random.seed(10)

# Generate Data
n = 125
x1 = np.random.randn(n)
x2 = x1 + 0.01 * np.random.randn(n)
correlation = np.corrcoef(x1, x2)[0, 1]
print(f"Correlation between x1 and x2: {correlation:.4f}")

y = 1 + 1*x1 + 2*x2 + 2*np.random.randn(n)

# Pairs Plot
fig, axs = plt.subplots(3, 3, figsize=(9, 9))

# Plotting x1 vs. x1, x2, y
axs[0, 0].hist(x1, bins=10, color='gray', edgecolor='black')
axs[0, 1].scatter(x1, x2, marker='.', color='blue')
axs[0, 2].scatter(x1, y, marker='.', color='blue')

# Plotting x2 vs. x1, x2, y
axs[1, 0].scatter(x2, x1, marker='.', color='blue')
axs[1, 1].hist(x2, bins=10, color='gray', edgecolor='black')
axs[1, 2].scatter(x2, y, marker='.', color='blue')

# Plotting y vs. x1, x2, y
axs[2, 0].scatter(y, x1, marker='.', color='blue')
axs[2, 1].scatter(y, x2, marker='.', color='blue')
axs[2, 2].hist(y, bins=10, color='gray', edgecolor='black')

# Setting axis labels
for i, label in enumerate(['x1', 'x2', 'y']):
    axs[i, 0].set_ylabel(label)
    axs[2, i].set_xlabel(label)

plt.tight_layout()
plt.show()
```





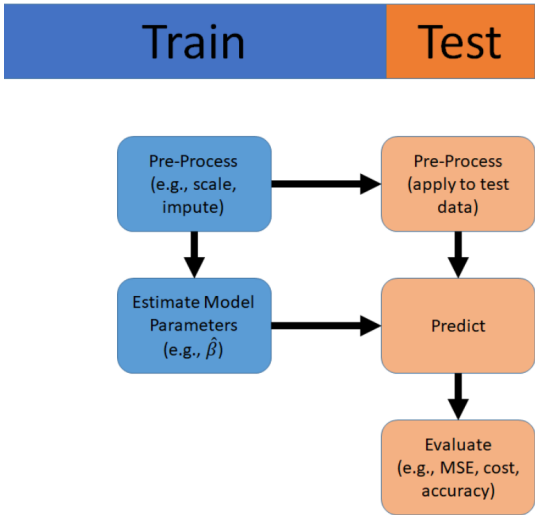
	True Coefficients	OLS	Ridge	Lasso
Intercept	1	1.214651	1.215677	1.222079
x1	1	18.528651	2.461695	2.974770
x2	2	-15.436997	0.613098	0.000000

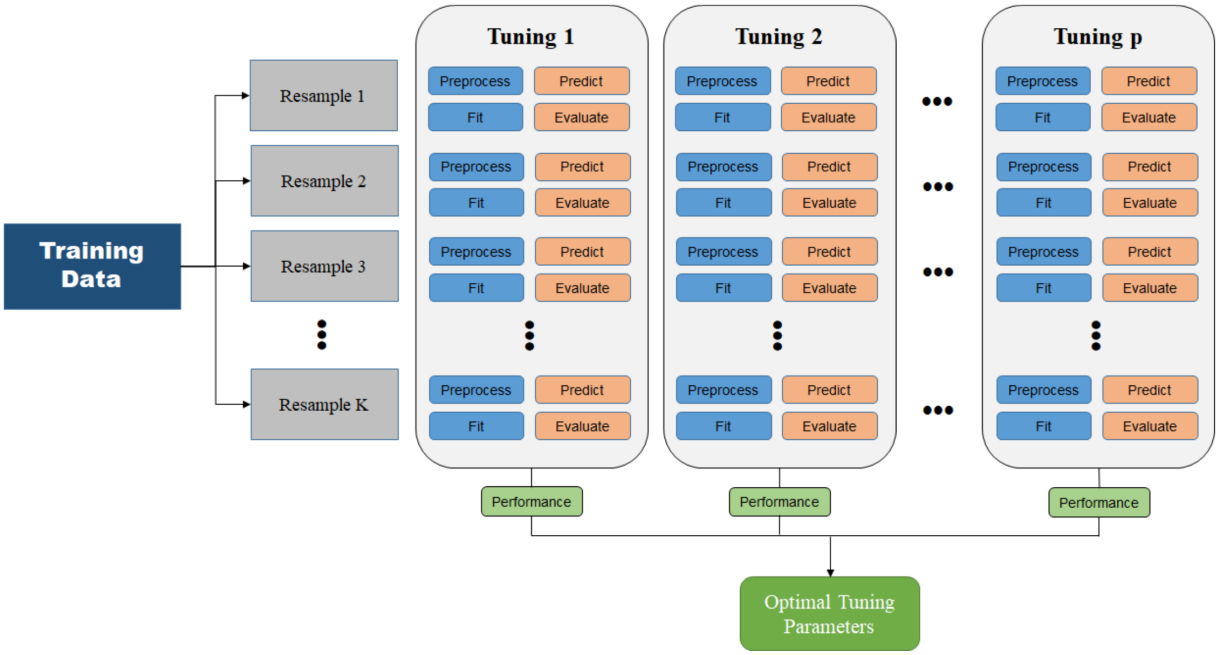
- Notice that the OLS coefficients have negative signs and large magnitude but a small constraint (penalty) produces a much closer result.
- That is, a small ridge or lasso penalty controlled the high variance.
- Ridge tends to shrink correlated predictors together.
- Lasso tends to choose one and set other(s)to zero.

5.5 Example with Strong Correlation

- Unlike ridge regression, the lass is *not* a linear smoother. There is no way to write $\hat{\boldsymbol{y}} = \mathbf{Hy}$.
- Thus, estimating the effective degrees of freedom is not based on trace of hat matrix.
- It turns out that the number of non-zero coefficients is a decent approximation of the effective number of parameters.
- We can use this value ($df = \sum_j 1(|\beta_j| > 0)$) in AIC/BIC/GCV for selecting λ.
    - Note: the *df* is not continuous in λ, so the min SSE model would have smallest λ within the set with $df = k$.

5.6 Cross-Validation and Penalized Regression





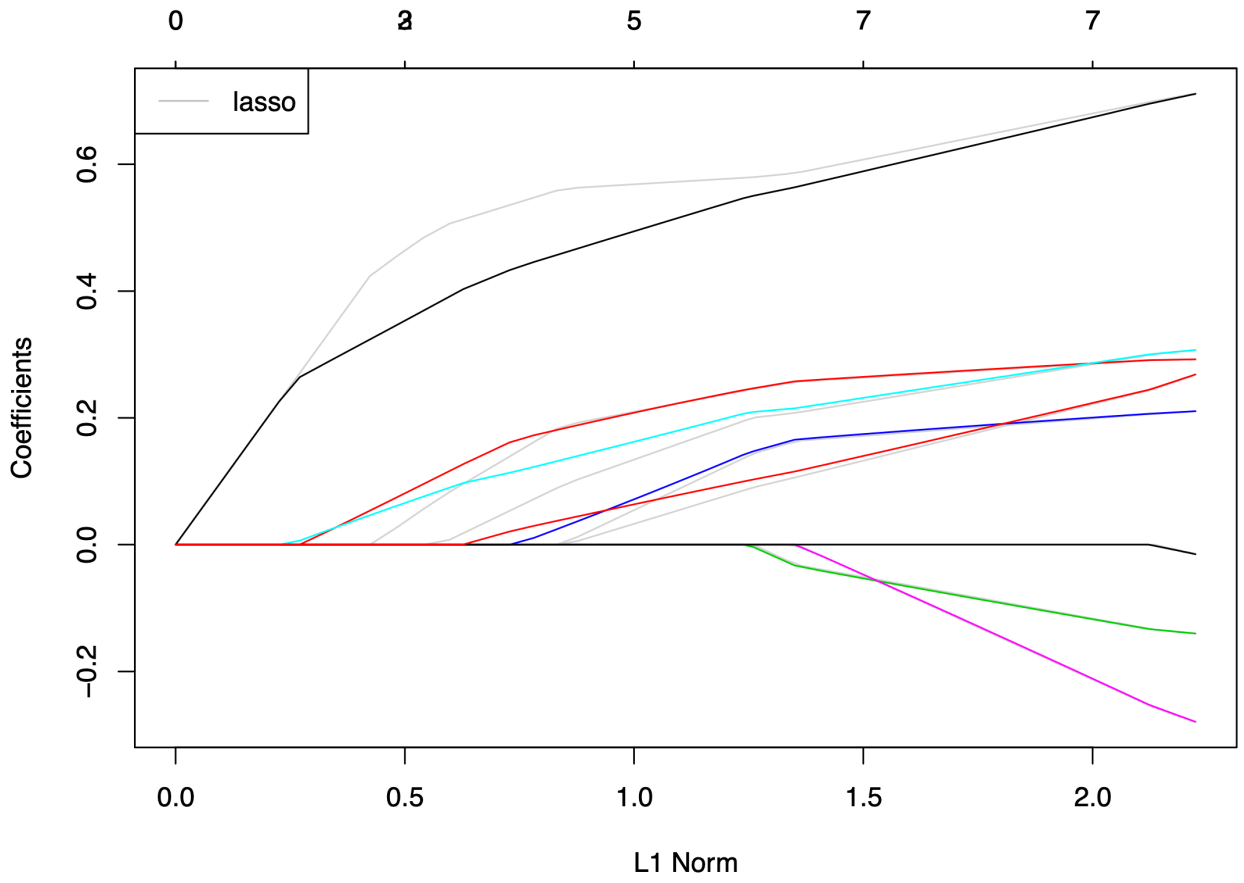
5.7 Elastic Net

The Elastic Net Penalty can help with selection (like lasso) and shrinks together correlated predictors (like ridge).

$$P(\beta, \alpha) = \sum_{j=1}^{p} \alpha\beta_j^2 + (1-\alpha)|\beta_j|$$

5.7.1 Comparing Elastic Net to Lasso and Ridge

Elastic Net with $\alpha = 0.5$.



5.7.2 Elastic Net Tuning

Notice that elastic net models have two *tuning parameters*: $\alpha \in [0,1]$ controls the type of penalty ($\alpha = 0$ for ridge, $\alpha = 1$ for lasso) and $\lambda \geq 0$ controls the strength of penalty.

There is not a shortcut (that I know of) for evaluating α; a basic loop over a grid of α values will work.

Note

There are few ways to jointly optimize α and λ.

1. Create a grid of α and λ values and fit models for every value. The problem is that the algorithm is optimized to search over a sequence of λ values in one pass. Also, due to the different penalties a reasonable λ sequence for a lasso penalty, it may not be reasonable for ridge penalty.
2. Use one resampling pass to estimate α, then another to estimate λ given α.
3. The chose of $\alpha > 0$ may not be very influential on performance, so may not have much practical need to optimize.

5.8 Relaxed Lasso

The relaxed lasso is basically an approach to use lasso for variable selection. The approach is as follows:

1. Use cross-validation to select λ.
2. Find the non-zero coefficients and select only those predictors to be in the model.
3. Fit an unpenalized (e.g., OLS) model using the selected features.

This attempts to use the lass variable selection to reduce variance, but unpenalized estimation to reduce bias in the model paramters.