9
Regularisation Techniques

Regularisation techniques are invaluable when dealing with complex datasets or situations where traditional methods may fall short. They are used to enhance model stability, improve predictive performance, and increase interpretability, especially when working with multi-dimensional data in multiple linear regression models and multivariate analyses. Regularisation addresses several common challenges in statistical modelling: i) multicollinearity, ii) variable selection, iii) overfitting, and iv) model simplification.

Environmental datasets often contain many independent variables, and it is likely that only some of them are necessary to explain the phenomenon of interest. **Variable selection** is the process of identifying the most important predictors to include in a model. This can be achieved through the application of specialist, domain-specific knowledge, or through statistical or data-driven approaches. Regularisation is an example of the latter, as it can automatically identify the most relevant predictors on statistical grounds, serving as an alternative to traditional variable selection methods such as Variance Inflation Factor (VIF) and stepwise selection (see Section 6.6.4 and Section 6.6.5).

Overfitting occurs when a model 'explains' the noise in data together with the underlying pattern, which might happen when the model has too many predictors relative to the number of observations. This may also result when variable selection has not been sufficiently addressed. An overfit model performs exceptionally well on training data but fails to generalise to new, unseen data. Additionally, having too many predictors can lead to **multicollinearity** (see Section 6.6.4). This is a common issue in multiple linear regression when some of the many predictors included in the model are correlated. Multicollinearity can lead to inflated standard errors, unstable coefficients, and difficulty interpreting the model. Regularisation help manage multicollinearity by shrinking coefficient estimates or setting some to zero.

Effectiveness in variable selection, reducing multicollinearity, and mitigating overfitting all contribute to **model simplification**. Regularisation achieves similar outcomes by shrinking coefficient estimates or setting some to zero, making the model easier to understand, explain, and interpret.

In this chapter, we will discuss three common regularisation techniques: Lasso, Ridge, and Elastic Net Regression.

9.1 RIDGE REGRESSION (L2 REGULARISATION)

Ridge regression mathematically 'tames' the wildness of linear regression when faced with multicollinearity. It achieves this by adding a penalty term to the linear regression loss function—a term proportional to the square of the coefficients (the L2 norm). This penalty nudges the coefficients towards zero, effectively shrinking them without forcing them to be exactly zero.

In linear regression, the loss function is typically the Mean Squared Error (MSE), which is the average of the squared residuals (also known as the residual sum of squares, RSS). The optimisation objective is to minimise this loss function. In other words, the linear regression model aims to find the coefficients that minimise the average squared difference between the observed values and the predicted values. The RSS is expressed in Equation 1:

$$RSS(\beta) = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 \tag{1}$$

And the MSE, which is the loss function to be minimised, is in Equation 2:

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

Where:

- y_i is the observed value for the i-th observation.
- β_0 is the intercept.
- β_j are the coefficients for the predictors.
- x_{ij} is the value of the j-th predictor variable for the i-th observation.
- n is the number of observations.
- p is the number of predictors.

The notation $RSS(\beta)$ and $MSE(\beta)$ indicates that these are functions of the coefficients β. The optimisation objective for linear regression is to find the coefficients β_0 and β_1 to β_p that minimise the MSE. This can be expressed in Equation Equation 3:

$$\min_{\beta}\left\{\frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2\right\} \tag{3}$$

Ridge regression extends the optimisation of the least squares regression by introducing a penalty term to the loss function. This penalty term is proportional to the square of the L2 norm of the coefficient vector, penalising large coefficient values. Ridge regression is specifically designed to handle multicollinearity and mitigate issues caused by correlated predictors. It also helps prevent overfitting when there are many predictors relative to the sample size, providing a more stable estimation process.

The penalty term is controlled by a hyperparameter[1] called lambda (λ) that determines the strength of the penalty. Larger values of λ lead to more shrinkage of

1. Hyperparameters are configuration settings that are external to your model and not learned from the data itself.

the coefficients. When $\lambda = 0$, Ridge Regression is equivalent to ordinary least squares regression. As λ approaches infinity, all coefficients (except the intercept) approach zero. To find the optimal λ, you might have to use techniques like cross-validation. Cross-validation will be discussed later in Section Section 9.4.

The loss function in Ridge Regression is given by Equation 4:

$$L_{ridge}(\beta) = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2 \tag{4}$$

Where λ is the regularisation parameter controlling the penalty's strength. Note that typically, the intercept β_0 is not included in the penalty term.

In Equation Equation 4, $L_{ridge}(\beta)$ is the Ridge Regression loss function. This loss function includes the residual sum of squares (RSS) plus a penalty term $\lambda \sum_{j=1}^{p}\beta_j^2$. The optimisation objective in Ridge Regression is to find the values of the coefficients β_1 through β_p that minimise this penalised loss function, while also finding the optimal value for the intercept β_0.

Ridge regression introduces a bias-variance trade-off. By shrinking the coefficients, it introduces a slight bias, as the model's predictions may not perfectly match the training data. However, this bias is often offset by a significant reduction in variance. The reduced variance means the model's predictions are more stable and less sensitive to small changes in the input data. This trade-off often results in improved overall predictive performance, especially on new, unseen data.

So, Ridge Regression sacrifices a bit of bias (accuracy on the sample data) to gain a lot in terms of reduced variance (generalisation to new data). This is a typical example of the bias-variance trade-off in statistical modelling and machine learning, where we often find that a bit of bias can lead to a much more robust and reliable model.

Unlike some other regularisation methods, such as principal component regression, Ridge Regression maintains the interpretability of the coefficients in terms of their relationship with the outcome. It is also versatile and can be applied to various types of regression models, including linear and logistic regression.

9.2 LASSO REGRESSION (L1 REGULARISATION)

Lasso (Least Absolute Shrinkage and Selection Operator) regression employs a different penalty term compared to Ridge Regression. Instead of squaring the coefficients, Lasso Regression takes their absolute values. The cost function in Lasso Regression is given in Equation Equation 5:

$$L_{lasso}(\beta) = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j| \tag{5}$$

In Equation Equation 5, $L_{lasso}(\beta)$ is the Lasso Regression loss function. It includes the residual sum of squares (RSS) plus a penalty term $\lambda \sum_{j=1}^{p}|\beta_j|$ (L1 norm). This penalty term is the sum of the absolute values of the coefficients, scaled by the regularisation parameter λ (similar to Ridge Regression). Lasso regression seeks

the values of β_0 through β_p that minimise $L_{lasso}(\beta)$. As with Ridge Regression, the intercept β_0 is typically not included in the penalty term.

The strength of Lasso Regression lies in its ability to shrink some coefficients all the way to zero, effectively eliminating those variables from the model. This automatic variable selection makes Lasso Regression well-suited for creating sparse models where only the most influential variables are retained. This simplification aids in interpretation and can enhance model performance by reducing noise and overfitting.

Lasso Regression still applies a degree of shrinkage for the coefficients that are not shrunk to zero. Shrinkage reduces their variance and provide more stable models that are less sensitive to fluctuations in the data. Similar to Ridge Regression, Lasso involves a trade-off between bias and variance. The shrinkage introduces a small bias but can greatly reduce variance and result in better overall predictions.

Lasso regression is useful when dealing with datasets that have a large number of potential predictor variables. It helps identify the most relevant predictors. The end results is a simpler and more interpretable model. If you suspect redundancy among your predictor variables, Lasso can prune them and retain only those that provide the best predictive value. As always, the optimal value for λ should be determined through techniques like cross-validation.

9.3 ELASTIC NET REGRESSION

Elastic net regression is a hybrid regularisation technique that combines the penalties of Ridge and Lasso Regression. It tries to provide the advantages of both methods and mitigate their drawbacks.

Here, the penalty term is the weighted average of the L1 (Lasso) and L2 (Ridge) penalties. A mixing parameter called alpha (α) controls the weighting between the two penalties. When $\alpha = 0$, Elastic Net is equivalent to Ridge Regression and when $\alpha = 1$ it is equivalent to Lasso Regression. For values of α between 0 and 1, Elastic Net blends the properties of both methods and provides some flexibility to regularisation.

The cost function in Elastic Net Regression is given in Equation 6:

$$L_{e_{net}}(\beta) = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\left(\alpha\sum_{j=1}^{p}|\beta_j| + (1-\alpha)\sum_{j=1}^{p}\beta_j^2\right) \tag{6}$$

Where α is the mixing parameter, with $0 \leq \alpha \leq 1$.

In Equation 6 there is the familiar RSS plus the combined penalty term that is a weighted sum of the L1 and L2 norms. The objective of Elastic Net Regression is again to minimise $L_{e_{net}}(\beta)$ by seeking optimal values of β_1 through β_p.

Like the other regularisation techniques, Elastic Net is also used when you have highly correlated predictors. While Lasso Regression might arbitrarily select one variable from a group and ignore the rest, Elastic Net tends to select groups of correlated features together and so provide a more comprehensive understanding of variable importance. The flexibility of adjusting the α parameter allows you to fine-tune the regularisation to best suit your specific dataset and modelling goals.

It balances variable selection (Lasso) and shrinkage (Ridge). Also, Elastic Net can outperform Lasso and Ridge Regression in terms of prediction accuracy when dealing with high-dimensional datasets where the number of predictors exceeds the number of observations.

Elastic net is a good option if you have a dataset with many potential predictor variables and suspect strong correlations among them. Use it when you are uncertain whether pure variable selection (Lasso) or pure shrinkage (Ridge) is the best approach. The challenge is that now we also have to tune the α parameter in addition to the regularisation parameter λ. A caveat is that Elastic Net retains the interpretability of individual coefficients but the interpretation becomes slightly more nuanced due to the mixed penalty term. This requires a thoughtful approach to understanding the model outputs.

9.4 CROSS-VALIDATION

The values of the hyperparameters (λ or α) significantly affect the model's performance and generalisation ability and so it necessitates careful optimisation. The `cv.glmnet()` function (see Section 9.5) automates this process by performing both hyperparameter tuning[2] and cross-validation. It systematically evaluates different combinations of λ or α values across multiple subsets of our data, using cross-validation to estimate their out-of-sample performance. This allows for the selection of the hyperparameter combination that yields the best performance and thus avoids the risk of overfitting and improves model generalisation.

The most widely used cross-validation method is k-fold cross-validation. The dataset is divided into k equally sized subsets (specified by the user). The subsets are called 'folds'. The model is then trained k times, each time using $k - 1$ folds for training and the remaining fold for validation. It provides a robust estimate of model performance by utilising all data points for both training and validation. It balances computational cost and bias reduction. But, the choice of k can influence results, and there's a trade-off between bias and variance: lower k values may lead to higher bias but lower variance, whilst higher k values do the opposite.

The general approach taken in k-fold cross validation is that, for each combination of hyperparameter values, we:

1. Perform k-fold cross-validation on the training data.
2. Calculate the average performance metric (e.g., mean squared error) across all folds.
3. Select the hyperparameter values that produced the best average performance.

This ensures that the hyperparameters we select are robust and generalissable to unseen data, rather than being overly influenced by the peculiarities of a single training set.

K-fold cross-validation is the most frequently-used form of cross-validation, but several other types exist. Some of them are:

2. The goal of hyperparameter tuning is to find the optimal combination of hyperparameters that leads to the best model performance on your specific dataset. This is done by systematically evaluating different hyperparameter values and selecting the combination that yields the best results.

Leave-one-out cross-validation (LOOCV) is an extreme case of k-fold cross-validation where k equals the number of data points. This method trains the model on all but one data point and validates on the left-out point, repeating this process for each data point. LOOCV provides an nearly unbiased estimate of model performance but can be computationally expensive for large datasets. It's most often used for small datasets where maximising training data is important. The downside is that LOOCV can suffer from high variance, especially for noisy datasets.

Stratified cross-validation ensures each fold maintains the same proportion of samples for each class as in the complete dataset. It useful for imbalanced datasets or when dealing with categorical outcomes. By preserving the class distribution in each fold, stratified cross-validation provides a more representative evaluation of model performance across all classes. Implementing stratification can be complex for multi-class problems or continuous outcomes.

Holdout validation is the simplest form of cross-validation. The dataset is split into a training set and a test set. Typically, about 70-80% of the data is used for training and the balance is reserved for testing. The model is trained on the training set and then evaluated on the held-out test set. It is computationally efficient and provides a quick estimate of model performance but it has several limitations. Firstly, because it doesn't make full use of the available data for training, it can be an issue for smaller datasets. Secondly, the results can be highly dependent on the particular split chosen, leading to high variance in performance estimates. This is especially true for smaller datasets or when the split doesn't represent the overall data distribution well. But holdout validation remains useful for large datasets or as a quick initial assessment before applying more complex cross-validation techniques.

The examples will show k-fold cross validation, but you can easily adapt the code to use other cross-validation methods.

9.5 R FUNCTION

In R, the **glmnet** package provides functions for fitting regularised linear models. The `cv.glmnet()` function performs cross-validated regularisation path selection for the Elastic Net, Lasso, and Ridge Regression models.

```r
cv.glmnet(x, y, alpha = 1, lambda = NULL, nfolds = 10,
          standardize = TRUE)
```

The function takes the following arguments:

- `x`: A matrix of predictors.
- `y`: A matrix of response variables (but read the help file as this varies depending on the data type).
- `alpha`: The mixing parameter for the Elastic Net penalty. When `alpha = 0`, the model is a Ridge Regression. When `alpha = 1`, the model is a Lasso Regression. The default value is `alpha = 1`.
- `lambda`: A vector of regularisation parameters. The function fits a model for each value of `lambda` and selects the best one based on cross-validation. The default

is `lambda = NULL`, which means the function will generate a sequence of `100` values between `10^-2` and `10^2`.
- `nfolds`: The number of folds in the cross-validation. The default is `nfolds = 10`.
- `standardize`: A logical value indicating whether the predictors should be standardised. The default is `standardize = TRUE`.

It is not clearly documented in the function's help file, but the 'glm' in the function name indicates that the function fits a generalised linear model. This implies 'gaussian,' 'binomial,' 'poisson,' 'multinomial,' 'cox,' and 'mgaussian' families are supported, which can be supplied via the `family` argument to the function. The 'net' part of the name indicates that the function fits an Elastic Net, thus allowing choose between Lasso and Ridge by setting `alpha` to 1 or 0 (or something in-between). The 'cv' part of the name indicates that the function performs cross-validation.

9.6 EXAMPLE 1: RIDGE REGRESSION

The data I use here should be well-known by now. They are the same seaweed dataset used throughout Chapter 6. I will use Ridge Regression to predict the response variable `Y` using the predictors `annMean`, `augMean`, `augSD`, `febSD`, and `febRange`.

First, I will read in the data and prepare them in the format required by `cv.glmnet()`. This involves standardising the response variable and predictors and converting them to matrices. I specify the range of λ values to try and set up 10-fold cross-validation. I then fit the model and plot the results of the cross-validation.

```
# Ridge Regression with Cross-Validation

# Set seed for reproducibility
set.seed(123)

# Load necessary libraries
library(glmnet)
library(tidyverse)

# Read the data
sw <- read.csv("data/spp_df2.csv")

# Standardise the response variable and present as a matrix
y <- sw ▷
  select(Y) ▷
  scale(center = TRUE, scale = FALSE) ▷
  as.matrix()

# Provide the predictors as a matrix
X <- sw ▷
  select(-X, -dist, -bio, -Y, -Y1, -Y2) ▷
  as.matrix()
```

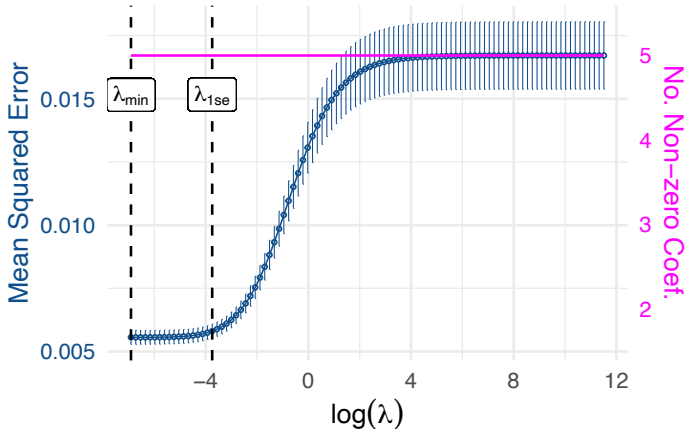

FIGURE 9.1. Cross-validation statistics for the Ridge Regression approach applied to the seaweed data.

```
# Set up lambda sequence
lambdas_to_try <- 10 ^ seq(-3, 5, length.out = 100)

# Perform 10-fold cross-validation
ridge_cv <- cv.glmnet(X, y, alpha = 0, lambda = lambdas_to_try,
  standardize = TRUE, nfolds = 10)
```

```
# Plot cross-validation results (ggplot shown)
plot(ridge_cv)
```

Figure 9.1, generated from the `cv.glmnet()` object, illustrates the relationship between the regularisation parameter λ and the model's cross-validation performance. The y-axis represents the mean squared error (MSE) from cross-validation, whilst the x-axis shows the $log(\lambda)$ values tested. Red dots indicate the mean MSE for each λ, with error bars showing ± 1 standard error. Two vertical dashed lines highlight important λ values: λ_{min}, which minimises the mean MSE, and λ_{1se}, the largest λ within one standard error of the minimum MSE. One may select the optimal λ using either the λ_{min} or the λ_{1se} rule, accessible via `cv.glmnet_object$lambda.min` and `cv.glmnet_object$lambda.1se`, respectively. To utilise the chosen λ, one refits the model using `glmnet()` and extract the coefficients.

For performance evaluation, one can calculate the sum of squared residuals (SSR) as the sum of squared differences between observed and predicted values, and the R-squared value as the square of the correlation between observed and predicted values, representing the proportion of variance in the dependent variable that is predictable from the independent variable(s).

The results show that the model explains 67.07% of the variance in the response

variable:

```
# Fit models and calculate performance metrics
fit_model_and_calculate_metrics <- function(X, y, lambda) {
  model <- glmnet(X, y, alpha = 0, lambda = lambda,
                  standardize = TRUE)
  y_hat <- predict(model, X)
  ssr <- sum((y - y_hat) ^ 2)
  rsq <- cor(y, y_hat) ^ 2
  list(model = model, ssr = ssr, rsq = rsq)
}

# Best cross-validated lambda
lambda_cv <- ridge_cv$lambda.min
mod_cv <- fit_model_and_calculate_metrics(X, y, lambda_cv)

# Print results
mod_cv
> $model
>
> Call:  glmnet(x = X, y = y, alpha = 0, lambda = lambda, standardize = TRUE)
>
>   Df  %Dev Lambda
> 1  5 67.06  0.001
>
> $ssr
> [1] 5.321994
>
> $rsq
>          s0
> Y 0.6706681
```

As already indicated, an alternative to using `lambda.min` for selecting the optimal λ value is to use the 1 SE rule, which is contained in the attribute `lambda.1se`. This reduces the risk of overfitting as it tends to select a simpler model. We can use this value to refit the model and extract the coefficients, as before.

AIC and BIC can also be used to select suitable models. These information criteria penalise the model for the number of parameters used, providing a balance between model complexity and goodness of fit. The `calculate_ic()` function below calculates the AIC and BIC for a given model and returns the results in a list. We can then use this function to calculate the AIC and BIC for each model fit with each λ in `lambdas_to_try`:

```
# Calculate AIC and BIC
calculate_ic <- function(X, y, lambda) {
  model <- glmnet(X, y, alpha = 0, lambda = lambda,
                  standardize = TRUE)
```

```
  betas <- as.vector(coef(model)[-1])
  resid <- y - (scale(X) %*% betas)
  H <- scale(X) %*%
    solve(t(scale(X)) %*% scale(X) + lambda *
            diag(ncol(X))) %*% t(scale(X))
  df <- sum(diag(H))
  log_resid_ss <- log(sum(resid ^ 2))
  aic <- nrow(X) * log_resid_ss + 2 * df
  bic <- nrow(X) * log_resid_ss + log(nrow(X)) * df
  list(aic = aic, bic = bic)
}

ic_results <- map(lambdas_to_try, ~ calculate_ic(X, y, .x)) ▷
  transpose()
```

A plot of the change in the information criteria with $log(\lambda)$ is shown in Figure 9.2. The optimal λ values according to both AIC and BIC can then be used to refit the model and arrive at the coefficients of interest.

```
# Plot information criteria
plot_ic <- function(lambdas, ic_results) {
  df <- data.frame(lambda = log(lambdas),
                   aic = unlist(ic_results$aic),
                   bic = unlist(ic_results$bic))

  df_long <- pivot_longer(df, cols = c(aic, bic),
                          names_to = "criterion",
                          values_to = "value")

  ggplot(df_long, aes(x = lambda, y = value, color = criterion)) +
    geom_line() +
    scale_color_manual(values = c("aic" = "orange", "bic" = "skyblue3"),
                       labels = c("aic" = "AIC", "bic" = "BIC")) +
    labs(x = "log(lambda)",
         y = "Information Criterion", color = "Criterion") +
    theme_minimal() +
    theme(legend.position = "top",
          legend.direction = "horizontal",
          legend.box = "horizontal")
}

plot_ic(lambdas_to_try, ic_results)
```

Now we find the λ values that minimise the AIC and BIC, and refit the models using these values. It so happens that both AIC and BIC selects the same λ values:

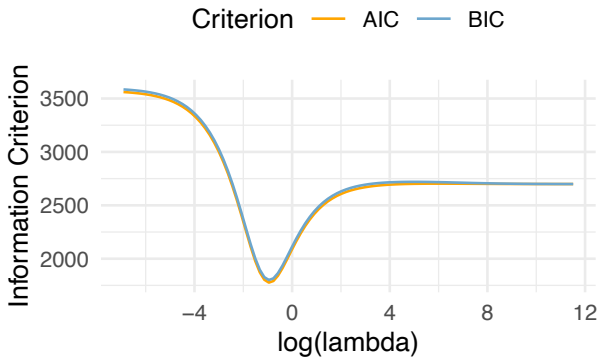

FIGURE 9.2. Plot of information criteria for best model fit selected through Ridge Regression.

```
# Optimal lambdas according to both criteria
lambda_aic <- lambdas_to_try[which.min(ic_results$aic)]
lambda_bic <- lambdas_to_try[which.min(ic_results$bic)]

# Fit final models using the optimal lambdas
mod_aic <- fit_model_and_calculate_metrics(X, y, lambda_aic)
mod_bic <- fit_model_and_calculate_metrics(X, y, lambda_bic)
```

For interest sake, we may also produce a plot that traces the coefficients of the model as λ changes. This can help us understand how the coefficients shrink as λ increases, and which variables are most important in the model. The plot below shows the Ridge Regression coefficients path for each variable in the model (Figure 9.3).

```
# Plot the Ridge Regression coefficients path
res <- glmnet(X, y, alpha = 0, lambda = lambdas_to_try,
              standardize = FALSE)
plot(res, xvar = "lambda")
legend("topright", lwd = 1, col = 1:6,
       legend = colnames(X), cex = 0.7)
```

So, after having demonstrated the different methods for selecting the optimal λ value, we can now summarise the results:

```
> [1] "CV Lambda: 0.001"
> [1] "AIC Lambda: 0.3854"
> [1] "BIC Lambda: 0.3854"
> [1] "CV R-squared: 0.6707"
> [1] "AIC R-squared: 0.6025"
> [1] "BIC R-squared: 0.6025"
```

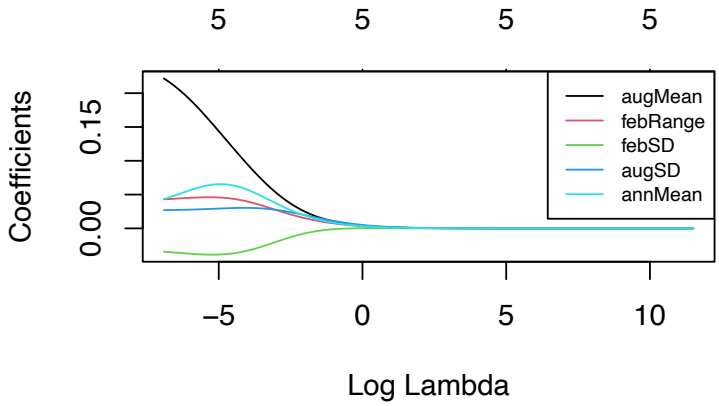

FIGURE 9.3. Plot of the Ridge Regression coefficients paths.

Now we can extract the coefficient produced from models selected via the AIC and CV methods.

```
res_aic <- glmnet(X, y, alpha = 0, lambda = lambda_aic,
                  standardize = FALSE)
res_aic
>
> Call:  glmnet(x = X, y = y, alpha = 0, lambda = lambda_aic, standardize = FALSE)
>
>   Df  %Dev Lambda
> 1  5 13.46 0.3854
coef(res_aic)
> 6 x 1 sparse Matrix of class "dgCMatrix"
>                       s0
> (Intercept) -0.021327121
> augMean      0.009856026
> febRange     0.007118466
> febSD       -0.001074341
> augSD        0.010696102
> annMean      0.008114467
```

```
res_cv <- glmnet(X, y, alpha = 0, lambda = lambda_cv,
                 standardize = FALSE)
res_cv
>
> Call:  glmnet(x = X, y = y, alpha = 0, lambda = lambda_cv, standardize = FALSE)
>
>   Df  %Dev Lambda
> 1  5 66.77  0.001
```

```
coef(res_cv)
> 6 x 1 sparse Matrix of class "dgCMatrix"
>                       s0
> (Intercept) -0.12384440
> augMean      0.22200994
> febRange     0.04287655
> febSD       -0.03446642
> augSD        0.02699458
> annMean      0.04324177
```

Ridge regression adds a penalty to the size of the coefficients, resulting in their shrinkage towards zero. This penalty affects all coefficients simultaneously. Notably, there is a difference in the model fit obtained using λ_{AIC} (which is larger) and λ_{min} (which is smaller). The former model explains 55.69% of the variance, compared to λ_{min}, which explains 63.37% of the variance.

Although shrinkage affects the absolute magnitude of the coefficients (they are biased estimates of the true relationships between the predictors and the response variable), the coefficients in Ridge Regression retain their general meaning—they still represent the change in the response variable associated with a one-unit change in the predictor variable, holding other predictors constant. While the absolute values of the coefficients may be biased due to regularisation, the relative importance of the predictors can still be interpreted. The magnitude of the coefficients can indicate the relative influence of each predictor on the response variable, even if their exact values are reduced.

Importantly, the predictive ability of the model can improve with shrunk coefficients because Ridge Regression reduces overfitting and enhances the model's generalisability to new, unseen data. By stabilising the coefficient estimates, the model often achieves better performance on validation and test datasets, which is important should robust predictive analytics be the goal.

9.7 EXAMPLE 2: LASSO REGRESSION

Doing a Lasso Regression is easy. Simply change the `alpha` parameter to 1 in the `glmnet` function. The rest of the code remains the same. I'll show only the final output of this analysis to avoid repetition.

```
# Print results
mod_cv
> $model
>
> Call:  glmnet(x = X, y = y, alpha = 1, lambda = lambda, standardize = TRUE)
>
>   Df %Dev Lambda
> 1  5   67  0.001
>
> $ssr
```

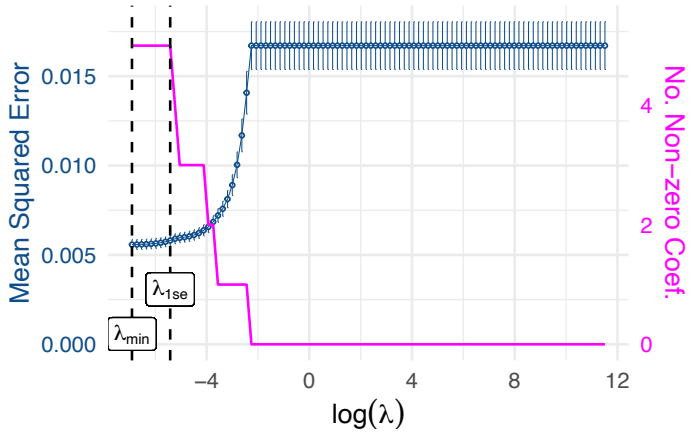

FIGURE 9.4. Cross-validation statistics for Lasso Regression applied to the seaweed data.

```
> [1] 5.332835
>
> $rsq
>           s0
> Y 0.6701255
coef(mod_cv$model)
> 6 x 1 sparse Matrix of class "dgCMatrix"
>                       s0
> (Intercept) -0.12886019
> augMean      0.26097296
> febRange     0.03431981
> febSD       -0.02497532
> augSD        0.02441380
> annMean      0.02021480

# Print results
print(paste("CV Lambda:", lambda_cv))
> [1] "CV Lambda: 0.001"
print(paste("CV R-squared:", round(mod_cv$rsq, 4)))
> [1] "CV R-squared: 0.6701"
```

Lasso regression incorporates an L1 penalty term in its cost function, which shrinks some coefficient estimates to exactly zero. By reducing certain coefficients to zero, Lasso effectively eliminates those predictors from the model, which achieves automatic variable selection:

- When λ is small, the penalty is minimal, and Lasso behaves similarly to ordinary least squares regression, retaining most coefficients.
- When λ is large, the penalty increases, causing more coefficients to shrink to zero.

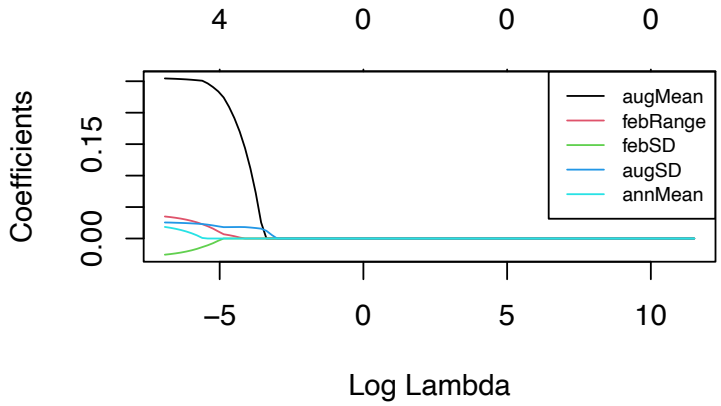

FIGURE 9.5. Plot of the Lasso Regression coefficients paths.

This results in a sparser model where only the most significant predictors have non-zero coefficients.

In our example (Figure 9.4), we see at λ_{min}, the number of non-zero coefficients is minimised—all five coefficients remain. At λ_{1se}, the number of non-zero coefficients decreases to four. Consequently, for higher values of λ, more predictors will have coefficients exactly equal to zero. This is also seen in Figure 9.4. In Figure 9.5 we can see that the first predictor to reach zero is `annMean`, then `febSD`, `febRange`, and so forth. The implication is that they are excluded from the model and the model is simplified. This leads to several benefits: reduced multicollinearity, improved interpretability, and better generalisation to new data.

Coefficients that remain non-zero after Lasso regularisation are considered more important predictors. Those remaining coefficients can be interpreted similarly to standard linear regression: as the expected change in the response variable for a one-unit change in the predictor, holding other predictors constant.

The λ parameter controls the amount of bias introduced. While Lasso can produce biased estimates, it reduces variance, often resulting in a model that performs better on new, unseen data. This trade-off enhances predictive accuracy but means that the exact coefficient values may not represent the true underlying relationships as closely as those in an unregularised model.

Despite regularisation, the relative magnitudes of the non-zero coefficients provide a glimpse into predictor importance. Larger absolute values of coefficients indicate stronger relationships with the response variable. The exact numerical values are biased, but ranking predictors by their coefficients still offers useful insight into their relative importance.

9.8 EXAMPLE 3: ELASTIC NET REGRESSION

In this last example we'll look at Elastic Net Regression, which combines the L1 and L2 penalties of Lasso and Ridge Regression. There are now two parameters to

optimise: α and λ. The α parameter controls the mix between the L1 and L2 penalties, with $\alpha = 0$ behaving like Ridge Regression and $\alpha = 1$ behaving like Lasso Regression. For α values between 0 and 1, Elastic Net combines the strengths of both Lasso and Ridge Regression. Optimisation of α and λ is also done using cross-validation. In practise, the steps are:

1. Set up a grid of α values (from 0 to 1) and λ values to try.
2. Performs cross-validation for each combination of α and λ using `cv.glmnet()`.
3. Select the best α and λ combination based on the minimum mean cross-validated error.
4. Fit the final model using the best α and λ.
5. Calculate the performance metrics.
6. For the Elastic Net model with the best alphaCreate plots similar to those in the Ridge and Lasso examples.

```r
# Define the range of alpha values to try
alphas_to_try <- seq(0, 1, by = 0.1)

# Define the range of lambda values to try
lambdas_to_try <- 10^seq(-3, 3, length.out = 100)

# Perform grid search with cross-validation
cv_results <- lapply(alphas_to_try, function(a) {
  cv.glmnet(X, y, alpha = a, lambda = lambdas_to_try,
            standardize = TRUE, nfolds = 10)
})

# Find the best alpha and lambda
best_result <- which.min(sapply(cv_results, function(x) min(x$cvm)))
best_alpha <- alphas_to_try[best_result]
best_lambda <- cv_results[[best_result]]$lambda.min

# Fit the final model with the best parameters
final_model <- glmnet(X, y, alpha = best_alpha,
                      lambda = best_lambda,
                      standardize = TRUE)

# Calculate performance metrics
y_hat <- predict(final_model, X)
ssr <- sum((y - y_hat) ^ 2)
rsq <- cor(y, y_hat) ^ 2
```

```
> [1] "Best Alpha: 0.3"
> [1] "Best Lambda: 0.001"
> [1] "R-squared: 0.6706"
```

The model coefficients are:

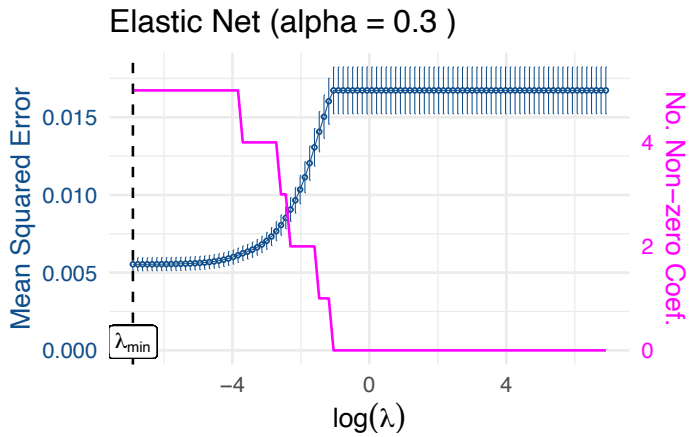

FIGURE 9.6. Cross-validation statistics for Elastic Net Regression applied to the seaweed data.

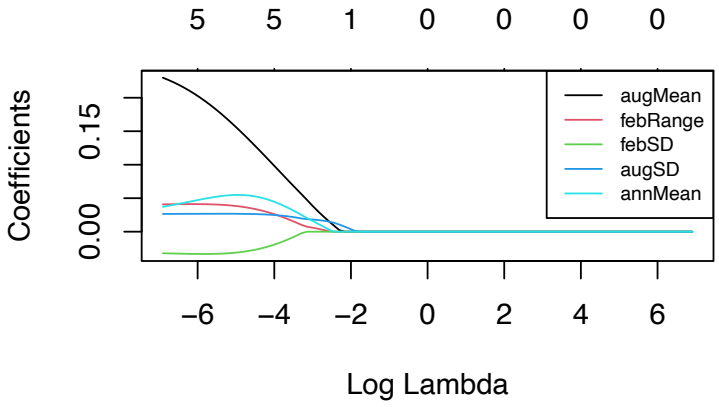

FIGURE 9.7. Plot of the Elastic Net Regression coefficients paths.

```
coef(cv_results[[best_result]])
> 6 x 1 sparse Matrix of class "dgCMatrix"
>                       s1
> (Intercept) -0.117063010
> augMean      0.240447330
> febRange     0.015404286
> febSD       -0.007224065
> augSD        0.014882111
> annMean      0.026504736
```

The interpretation of coefficients in Elastic Net is a blend of Ridge and Lasso. Some coefficients may be shrunk to zero (feature selection), while others are shrunk but remain non-zero (magnitude reduction). The non-zero coefficients retain their general meaning with an emphasis on their relative importance.

9.9 THEORY-DRIVEN AND DATA-DRIVEN VARIABLE SELECTION

The choice between theory-driven and data- or statistics-driven variable selection represents an important consideration that can greatly influence model interpretation, its predictive power, and your value as an ecologist. This decision reflects a broader tension in scientific methodology between deductive and inductive reasoning. Each offers advantages and limitations that you should be aware of as an ecologist.

Theory-driven variable selection is core to the scientific method. It relies on *a priori* knowledge and established ecological theories (as far as they exist in ecology!) to guide your choice of predictors in a model. This aligns closely with the hypothetico-deductive method, where we formulate hypotheses based on existing knowledge and subsequently test these against the data we collect. The strength of this method lies in its interpretability. Models built on theoretical foundations often contribute directly to testing and refining ecological hypotheses. By focusing on variables with known or hypothesised relationships (with mechanisms often rooted in ecophysiological or ecological inquiries), the theory-driven hypothetico-deductive method should lead to more parsimonious models that are less prone to overfitting and more reflecting of reality.

Theory-driven selection is not without its drawbacks. It requires that we have a good grasp of the mechanism underlying our favourite ecological system. This is not always the case in complex systems where the underlying mechanisms are not well understood. Theory-driven selection can then lead to the exclusion of important variables that were not initially hypothesised and it can limit the scope of the analysis and potentially overlook significant relationships in the data.

A naive young ecologist might place undue value on the notion that their hard work collecting diverse data and developing hypotheses should all be reflected in their final model. This can lead to confirmation bias, where one is more likely to select variables that support our hypotheses and ignore those that do not. This bias can compromise the objectivity of the model and lead to skewed results that do not accurately represent the underlying ecological processes.

Moreover, the insistence on including all variables that were initially considered important can result in overly complex models. Such models can be difficult to interpret and may suffer from overfitting, where the model captures noise rather than the true signal in the data. Overfitted models perform well on the data we collected but poorly on new, unseen data. The consequence is a loss of predictive power and generalisability.

Another weakness of theory-driven variable selection is that the reliance on existing theories or the novel, promising hypothsis of the day may lead us to overlook important but unexpected relationships in the data. In complex ecological systems, where our theoretical understanding may be incomplete, some variables could be missed entirely—these might in fact hold the key to the real cause of the ecological patterns we observe. This limitation becomes concerning when studying ecosystems or phenomena that are not well understood or are undergoing rapid changes, such as those affected by climate change or novel anthropogenic pressures.

On the other hand, data-driven approaches, including regularisation techniques, VIF, and forward model variable selection (Chapter 6), allow the data itself to guide variable selection. These methods are increasingly used in today's era of high-dimensional datasets common in modern ecological research. The primary advantage of data-driven selection lies in its potential for discovery—it can uncover unexpected relationships and generate new hypotheses, which is valuable in complex ecological systems where interactions may not be immediately apparent.

Data-driven methods are well-suited for handling the complexity often encountered in environmental and ecological datasets, where numerous potential predictors may co-occur and interact. They offer a degree of objectivity, reducing the potential for our personal biases in variable selection. But these approaches are not without risks. There's a danger of identifying relationships that are statistically significant but ecologically meaningless—we refer to this as spurious correlations (e.g. the belief that consuming carrots significantly improves our night vision). Moreover, models with many variables can present significant interpretability challenges, especially when complex interactions are present. This can make it difficult to extract meaningful (plausible) insights from the model and to communicate results to a broader audience.

In practice, the most robust approach to selecting which of the multitude of variables to include in our model often involves a thoughtful combination of theory-driven and data-driven methods. Well-trained ecologists should start with theory-driven variable selection to identify the core predictors based on established ecological principles. We could then employ regularisation techniques to explore additional variables and potential interactions, and use the results to refine our models and generate new hypotheses for future research.

This hybrid approach combines the strengths of both methods. It allows for rigorous hypothesis testing while remaining open to unanticipated and new insights from the data. In ecology, where systems are often characterised by complex, non-linear relationships and interactions that may vary across spatial and temporal scales, this two-pronged approach offers distinct benefits.

Consider how these methods complement theoretical knowledge. Use variable selection methods as tools for prediction, and to assit generating new insights and

hypotheses about ecosystems. The choice between theory-driven and data-driven variable selection is not a binary one, but rather a spectrum of approaches.