Lecture 7

Penalty Functions

Penalized regression techniques are used when traditional linear regression models face challenges like multicollinearity or overfitting. By introducing penalty functions, these models balance fitting the data well and maintaining model simplicity. In this lecture, we'll examine penalty functions, their pros and cons, and other issues. We'll concern ourselves with optimization in the next lecture.

In traditional penalized regression, we have an objective: minimize the loss function (typically sum of squared errors) while introducing a penalty for large coefficients.

Common Types of Penalties:

- L1 (LASSO): Adds an absolute value penalty.
- L2 (Ridge): Adds a squared value penalty.
- Elastic Net: A combination of L1 and L2 penalties.

Penalty functions can also be used in spline regression.

Formulation of Penalties

Ridge Regression (L2 Penalty):

Objective: Minimize $\sum_{i=1}^{n} \frac{1}{n}(y_i - \hat{y}_1)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$

Penalty: $\lambda \sum_{j=1}^{p} \beta_j^2$

Effect: Shrinks coefficients towards zero but does not set any coefficient exactly to zero.

LASSO Regression (L1 Penalty)

Objective: Minimize $\sum_{i=1}^{n} \frac{1}{n}(y_i - \hat{y}_1)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$

Penalty: $\lambda \sum_{j=1}^{p} |\beta_j|$

Effect: Can shrink some coefficients to zero, effectively performing variable selection.

Elastic Net

Objective: Combine the strengths of Ridge and LASSO by minimizing $\sum_{i=1}^{n} \frac{1}{n}(y_i - \hat{y}_1)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$

Penalty: $\sum_{i=1}^{n} \frac{1}{n}(y_i - \hat{y}_1)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$

Effect: Handles cases where predictors are highly correlated and can perform both shrinkage and variable selection.

Splines in Regression

Use: Introduce flexibility in regression models by using piecewise polynomials.

Penalized Splines: Control smoothness by penalizing the roughness of the spline curve

Implementing Ridge Regression

```
ridge_regression <- function(X, y, lambda) {
 # Adding intercept term
 X <- cbind(1, X)
```

```
  # Identity matrix
  I <- diag(ncol(X))
  I[1, 1] <- 0  # Don't penalize the intercept

  # Normal equation with ridge penalty
  beta_hat <- solve(t(X) %*% X + lambda * I) %*% t(X) %*% y
  return(beta_hat)
}

# Example usage with mtcars dataset
X <- as.matrix(mtcars[, c("wt", "hp", "qsec")])
y <- mtcars$mpg
lambda <- 1

ridge_coefficients <- ridge_regression(X, y, lambda)
print(ridge_coefficients)
```

In this example, we've specified a λ scaling value, and have used matrix multiplication to solve the system rather than to use an optimization strategy.

Implementing LASSO regression

```
lasso_regression <- function(X, y, lambda, max_iter = 1000, tol = 1e-4) {
  n <- nrow(X)
  p <- ncol(X)
  beta <- rep(0, p)
  for (iter in 1:max_iter) {
    beta_old <- beta
    for (j in 1:p) {
      X_j <- X[, j]
      residual <- y - X %*% beta + beta[j] * X_j
      rho <- sum(X_j * residual)
      if (rho < -lambda / 2) {
        beta[j] <- (rho + lambda / 2) / sum(X_j^2)
      } else if (rho > lambda / 2) {
        beta[j] <- (rho - lambda / 2) / sum(X_j^2)
      } else {
        beta[j] <- 0
      }
    }
    if (sum(abs(beta - beta_old)) < tol) break
  }
  return(beta)
}

# Example usage with mtcars dataset
X <- as.matrix(mtcars[, c("wt", "hp", "qsec")])
X <- scale(X)  # Standardize predictors
```

```
y <- scale(mtcars$mpg)
lambda <- 0.1

lasso_coefficients <- lasso_regression(X, y, lambda)
print(lasso_coefficients)
```

In this case, we aren't able to use a direct matrix approach to the minimization, so an optimization algorithm is required. This means that LASSO regression is computationally more intensive and will take longer to implement, especially if there are a lot of variables, than will Ridge regression.

Elastic Net

```
elastic_net <- function(X, y, alpha, lambda, max_iter = 1000, tol = 1e-4) {
  # Decompose the elastic net into a combination of ridge and lasso
  ridge_part <- (1 - alpha) * ridge_regression(X, y, lambda)
  lasso_part <- alpha * lasso_regression(X, y, lambda, max_iter, tol)

  # Combine the results
  beta_hat <- ridge_part + lasso_part
  return(beta_hat)
}

# Example usage with mtcars dataset
alpha <- 0.5  # 50% ridge, 50% lasso
elastic_net_coefficients <- elastic_net(X, y, alpha, lambda)
print(elastic_net_coefficients)
```

	Pros	**Cons**
Ridge Regression	Useful when dealing with multicollinearity. Does not eliminate variables (all coefficients are shrunken but non-zero).	Cannot perform variable selection (all predictors stay in the model). Harder to interpret when many variables are included.
LASSO Regression	Performs variable selection (can shrink some coefficients to zero). Leads to simpler and interpretable models.	Can be unstable when predictors are highly correlated. Not as effective in situations where multicollinearity is present.
Elastic Net	Combines the strengths of Ridge and LASSO. Performs well with correlated predictors.	Introduces an additional tuning parameter (alpha).

Tuning Parameters

Lambda: Controls the strength of the penalty.

- Higher lambda leads to more regularization (larger penalty).
- Lower lambda leads to less regularization (smaller penalty).

Alpha (for Elastic Net): Balances between L1 and L2 penalties.

Feature Scaling

- Importance: Penalized regression assumes predictors are on the same scale.
- Solution: Always standardize/normalize predictors before applying penalized regression.

Penalized regression is one of the regression models where scaling makes a big difference in performance, a behaviour we usually expect from classification models.

```
# Example: Standardizing predictors
X <- scale(mtcars[, c("wt", "hp", "qsec")])
```

Note that in this example, the code uses the built-in scale function. This is something that you can customize, however, it's not the focus of this section. If we were to introduce validation with test and training sets, how would the scaling function lead to potential problems?

Spline Regression

Spline regression is a flexible method used to model non-linear relationships by dividing the data into segments and fitting simple models (typically polynomials) within each segment. To avoid overfitting and to ensure smooth transitions between segments, penalty functions are often incorporated into spline regression models.

Understanding Splines:

- Splines are piecewise polynomials used to model data, with each piece called a "knot."
- Knots are the points where the data is divided. More knots mean more flexibility but also a higher risk of overfitting. 3-5 knots is common, but the exact number, and how they are selected, can depend on the number of observations.
- Cubic splines are a common type, using cubic polynomials between each pair of knots, however, lower degrees are possible, including linear segments (cubics are preferred because they are smoother and create fewer continuity issues at the join points).

Challenges with Splines:

Without penalties, splines can fit the training data too closely, leading to overfitting. To control the smoothness and complexity of the spline, penalties are introduced.

Penalized Spline Regression

In penalized spline regression, a penalty is added to the fitting criterion (often the residual sum of squares) to discourage the spline from being too "wiggly."

Spline Regression Objective Function:

For a given set of data points (x_i, y_i), the objective in spline regression is to minimize the following:

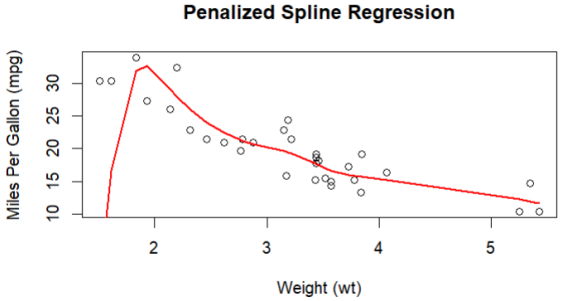
Minimize $\sum_{i=1}^{n} \frac{1}{n} \left(y_i - f(x_i)\right)^2 + \lambda \int \left(f''(x)\right)^2 dx$

- $f(x)$ is the spline function.
- The first term is the residual sum of squares, measuring the fit of the model to the data.
- The second term is a penalty on the roughness of the spline (i.e., the integral of the squared second derivative of $f(x)$.
- λ is the smoothing parameter that controls the trade-off between fitting the data and the smoothness of the spline. A larger λ results in a smoother spline.

Penalty Function:

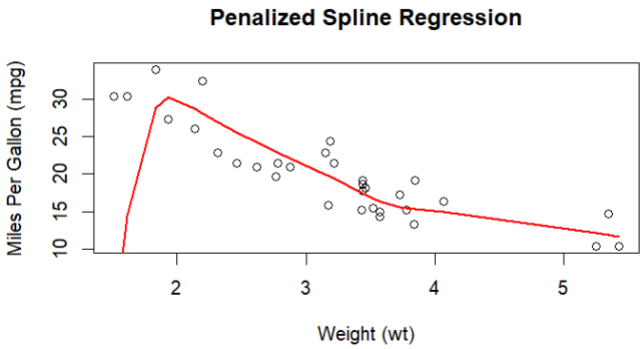
The penalty function $\lambda \int \left(f''(x)\right)^2 dx$ measures the roughness of the spline. A higher value indicates a more wiggly or complex spline. The goal is to balance fitting the data well and keeping the spline smooth. This balance is controlled by the smoothing parameter λ.

```
# Load required libraries
library(splines)

# Create a function for penalized spline regression
penalized_spline <- function(x, y, lambda) {
 # Create B-spline basis
 spline_basis <- bs(x, degree = 3, df = 10) # cubic spline with 10 degrees of freedom

 # Set up the penalty matrix (penalize the second derivative)
 penalty_matrix <- diff(diag(ncol(spline_basis)), differences = 2)

 # Estimate the coefficients using penalized least squares
 beta <- solve(t(spline_basis) %*% spline_basis + lambda * t(penalty_matrix) %*%
penalty_matrix) %*% t(spline_basis) %*% y

 # Return the fitted values
 fitted_values <- spline_basis %*% beta
 return(list(fitted_values = fitted_values, beta = beta))
}

# Apply the penalized spline to the mtcars dataset
x <- mtcars$wt
y <- mtcars$mpg
lambda <- 1

result <- penalized_spline(x, y, lambda)

# Plot the results
plot(x, y, main = "Penalized Spline Regression", xlab = "Weight (wt)", ylab = "Miles Per Gallon
(mpg)")
lines(sort(x), result$fitted_values[order(x)], col = "red", lwd = 2)
```

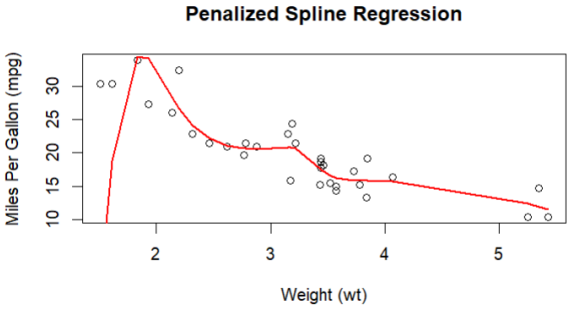


The example uses $\lambda = 1$, but let's also look at what happens if we increase or decrease λ.

If $\lambda = 10$, we get:



If $\lambda = 0.1$, we get:



You can see that the difference between 1 and 10 here is relatively small in terms of the smoothness, but when we reduce λ, we can see that we do get more wobbliness. As with other models with hyperparameters, experimentation may be required to select the best value.

Creating the B-Spline Basis:

The `bs()` function is used to create a B-spline basis matrix with cubic splines. You can control the degrees of freedom (df) to manage the flexibility of the spline.

Penalty Matrix:

A penalty matrix is created to penalize the second derivative of the spline. The `diff()` function is used to compute differences, effectively creating the penalty for the roughness of the spline.

Estimation Using Penalized Least Squares:

The coefficients β are estimated by solving a penalized least squares problem, which balances fitting the data and penalizing the roughness.

Plotting the Results:

The fitted spline is plotted against the original data to visualize the effect of the penalty. The fitted spline values can be further used to calculate the residuals and any metrics to be applied.

Scaling Issues:

- Tuning λ: The choice of λ can be customized based on the dataset. Common methods include cross-validation.
- Feature Scaling: As with other penalized regression techniques, predictors should be scaled before applying splines to avoid issues with different scales affecting the penalty. (This is not generally an issue when there is only one variable, but will matter if there are two or more

variables. Consider what happens if you cube an already large number and consider checking the consequences of rescaling or shifting variables.)

Pros and Cons of Splines:

	Pros	Cons
Splines (B-splines)	*Flexibility*: Splines can capture non-linear relationships in data. *Control over Smoothness*: The penalty function allows control over the smoothness of the fitted spline. *Interpretability*: Depending on the choice of knots and smoothing, splines can offer a balance between flexibility and interpretability.	*Choice of Smoothing Parameter*: Selecting the appropriate λ is critical and often requires cross-validation or other tuning methods. *Computational Complexity*: As the number of knots increases, the computation becomes more intensive. *Risk of Overfitting*: Without appropriate penalization, splines can overfit the data, especially with a large number of knots.

Splines can be extended to handle more than one variable. When splines are applied to multiple variables, they are typically referred to as **multivariate splines** or **tensor product splines**. These splines are used to model complex relationships involving two or more predictors and allow the model to capture interactions between these variables.

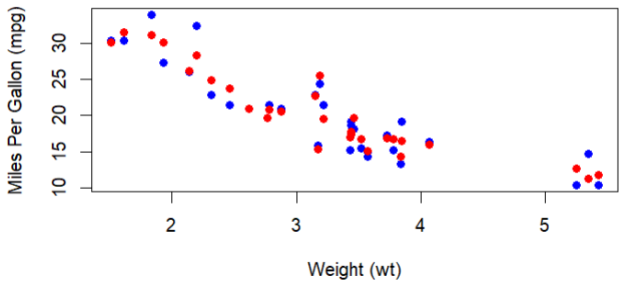
Multivariate Splines:
When working with multiple variables, splines can be extended in the following ways:

- Additive Splines: Each predictor is modeled with its own univariate spline, and the results are summed to create the final model. This is a common approach used in generalized additive models (GAMs).
- Tensor Product Splines: These are used when you want to model interactions between multiple predictors. Tensor product splines involve taking the Cartesian product of the basis functions for each predictor, creating a more flexible model that can capture interactions.

Let's look at some examples:
Multivariate Splines in R (Additive Splines)
Let's look at an example where we apply additive splines to model the relationship between mpg and two predictors (wt and hp) from the mtcars dataset.

```
# Load required library
library(splines)

# Generate B-spline basis functions for two variables
spline_basis_wt <- bs(mtcars$wt, degree = 3, df = 5)  # Spline for weight
```

```
spline_basis_hp <- bs(mtcars$hp, degree = 3, df = 5)  # Spline for horsepower

# Combine the spline basis functions into a design matrix
design_matrix <- cbind(spline_basis_wt, spline_basis_hp)

# Fit a linear model using the design matrix
model <- lm(mtcars$mpg ~ design_matrix)

# Summarize the model
summary(model)

# Generate predictions and plot
predicted_mpg <- predict(model)

# Plot the original data and fitted values
plot(mtcars$wt, mtcars$mpg, pch = 19, col = 'blue', xlab = 'Weight (wt)', ylab = 'Miles Per Gallon (mpg)')
points(mtcars$wt, predicted_mpg, col = 'red', pch = 19)
```




```
Call:
lm(formula = mtcars$mpg ~ design_matrix)

Residuals:
    Min      1Q  Median      3Q     Max
-2.8453 -1.2367  0.0648  0.5010  4.1152

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      33.6587     2.0706  16.256 2.26e-13 ***
design_matrix1   -9.8289     4.9036  -2.004  0.05809 .
design_matrix2   -8.6508     2.8536  -3.032  0.00635 **
design_matrix3  -13.5793     3.6646  -3.706  0.00131 **
design_matrix4  -14.8633     7.7267  -1.924  0.06806 .
design_matrix5  -16.9546     2.9589  -5.730 1.09e-05 ***
design_matrix1    7.5475     4.9986   1.510  0.14597
design_matrix2   -8.4160     2.5922  -3.247  0.00386 **
design_matrix3    0.2423     4.9980   0.048  0.96179
design_matrix4  -11.9081     3.9036  -3.051  0.00608 **
design_matrix5   -6.5238     3.1612  -2.064  0.05162 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.078 on 21 degrees of freedom
Multiple R-squared:  0.9195,   Adjusted R-squared:  0.8811
F-statistic: 23.98 on 10 and 21 DF,  p-value: 2.813e-09
```

Basis Functions for Each Predictor:
The `bs()` function is used to create B-spline basis functions for `wt` and `hp`. The `degree` parameter controls the degree of the splines (e.g., cubic splines with `degree = 3`). The `df` parameter controls the degrees of freedom, which effectively controls the number of knots.

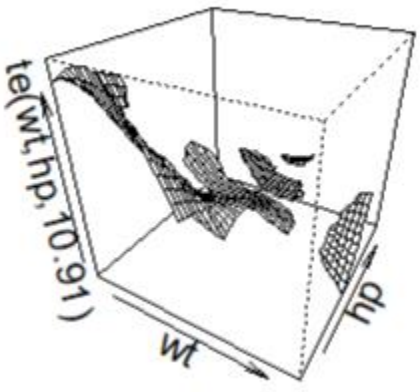
Combining Basis Functions:
The basis functions for `wt` and `hp` are combined into a single design matrix.

Fitting the Model:
A linear model is fitted using the design matrix. This model effectively uses additive splines for both predictors.

Plotting the Results:
The fitted values are plotted against the original data to visualize the model's fit.

Tensor Product Splines:
For modeling interactions between multiple variables, tensor product splines are more appropriate. In R, the `mgcv` package provides a straightforward way to fit tensor product splines through the `gam()` function.

```
# Load mgcv library for generalized additive models (GAMs)
library(mgcv)

# Fit a GAM with tensor product splines for wt and hp
gam_model <- gam(mpg ~ te(wt, hp), data = mtcars)

# Summarize the model
summary(gam_model)

# Plot the smooth interaction between wt and hp
plot(gam_model, scheme = 1)
```



```
Family: gaussian
Link function: identity

Formula:
mpg ~ te(wt, hp)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  20.0906     0.3275   61.34   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
           edf Ref.df     F p-value
te(wt,hp) 10.91  12.74 23.83  <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.905   Deviance explained = 93.9%
GCV = 5.4686  Scale est. = 3.433     n = 32
```

Tensor Product Spline:
The `te()` function in `gam()` is used to specify a tensor product spline, which models the interaction between `wt` and `hp`.

Plotting:
The `plot()` function allows you to visualize the smooth interaction between the two variables.

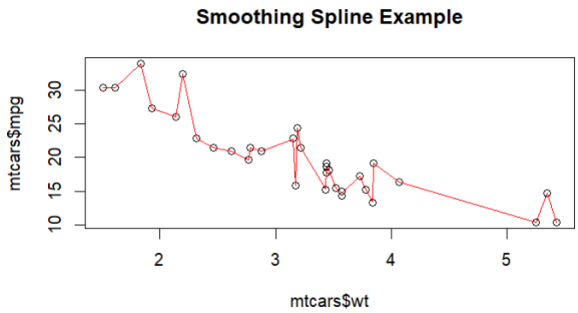
Splines are a versatile tool in regression modeling, and several different types of splines can be used depending on the specific needs of the modeling task. While B-splines are one of the most commonly used, other types of splines, such as smoothing splines, I-splines, N-splines, and others, are also available and can be used under different circumstances. Here's an overview of some of these types and their usage in regression modeling:

Smoothing Splines
Overview: Smoothing splines are a type of spline regression that automatically selects the number and placement of knots based on a smoothing parameter, often chosen by cross-validation. Instead of specifying knots manually, the model tries to balance fitting the data closely and maintaining smoothness.

Usage in Regression:
Purpose: Smoothing splines are used when you want a smooth curve that fits the data, but without overfitting.
How They Work: They minimize a loss function that includes both the residual sum of squares and a penalty for roughness (similar to penalized splines).
Formula: Minimize $\sum_{i=1}^{n} \frac{1}{n}\left(y_i - f(x_i)\right)^2 + \lambda \int \left(f''(x)\right)^2 dx$
In R: You can fit smoothing splines using the smooth.spline() function in R.
Limitations: One drawback is that it assumes a single smoothing parameter, which may not be ideal for all sections of the data.

```
model <- smooth.spline(mtcars$wt, mtcars$mpg)
plot(mtcars$wt, mtcars$mpg, main="Smoothing Spline Example")
```

```
lines(model, col="red")
```



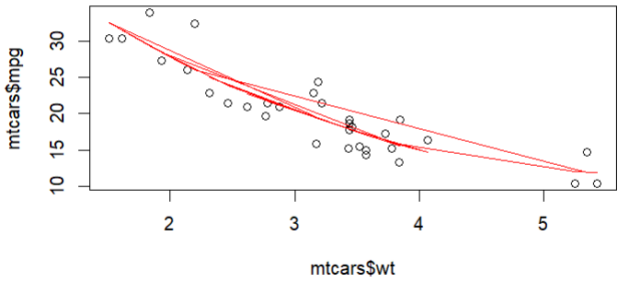
I-Splines (Integral Splines)
Overview: I-splines are a form of non-negative splines that are used primarily in the context of monotonic regression, where you want the regression function to be increasing or decreasing.

Usage in Regression:
Purpose: Useful in situations where prior knowledge dictates that the relationship between predictors and the response should be monotonic.
How They Work: I-splines are integrated from M-splines, which are piecewise polynomial functions. The integration process ensures that I-splines are non-decreasing.
Limitations: Not widely used in general-purpose regression because they impose monotonicity constraints, which may not be suitable for many applications.
In R: I-splines are less commonly implemented in standard packages, but custom implementations or extensions might be available in more specialized libraries.

N-Splines (Natural Splines)
Overview: Natural splines are a type of spline where the endpoints (knots at the boundaries of the data) are forced to be linear, which reduces the risk of overfitting at the extremes.

Usage in Regression:
Purpose: Used when you want a smoother and more interpretable spline model, especially at the boundaries of your data.
How They Work: N-splines impose a linear constraint on the endpoints of the spline, ensuring that the function does not exhibit erratic behavior near the boundaries.
In R: Natural splines can be implemented using the ns() function from the splines package.
Limitations: While N-splines offer more stability at the boundaries, they might not capture extreme non-linearity as effectively as unconstrained splines.

```
library(splines)
model <- lm(mpg ~ ns(wt, df=4), data=mtcars)
plot(mtcars$wt, mtcars$mpg)
lines(mtcars$wt, predict(model), col="red")
```



Note: you may want to sort your data in your dataframe in terms of the input variable.

Other Splines (e.g., P-splines, T-splines)

P-Splines (Penalized Splines): P-splines are similar to smoothing splines, but they explicitly incorporate a penalty function for the roughness of the spline. They allow for more flexibility in controlling smoothness by using a combination of B-splines and a penalty term. These are often used in generalized additive models (GAMs) to allow for more complex, yet smooth, relationships.

T-Splines (Tensor Splines): Tensor product splines are used when dealing with multidimensional data, particularly when interactions between variables need to be modeled smoothly. They are often used in spatial data analysis or when modeling surfaces.

Comparison and Limitations

- *B-Splines* are versatile and commonly used, particularly in their unpenalized form in standard regression or with penalties in GAMs.
- *Smoothing Splines* are popular for their simplicity in automatically balancing fit and smoothness but might be limited in very high-dimensional or highly non-linear contexts.
- *I-Splines* are specialized for monotonic regression and are less commonly used outside of this context.
- *N-Splines* are useful for ensuring more stable behavior at the boundaries of the data, making them preferable in certain regression tasks where boundary behavior is critical.

Different types of splines serve different purposes in regression modeling. While some, like B-splines and smoothing splines, are widely used in general-purpose regression, others like I-splines and N-splines serve more specialized roles. Understanding the specific properties and use cases of each type of spline is important for effectively applying them to real-world data modeling tasks.

Resources:

1. https://faculty.washington.edu/yenchic/18W_425/Lec11_Reg03.pdf
2. https://www.cs.cmu.edu/afs/cs/academic/class/15462-s10/www/lec-slides/lec06.pdf