Lecture 11

Variable Selection in Multiple Regression

Let's review.

Variable selection is a crucial step in multiple regression modeling, as it helps identify the most relevant predictors and improve model performance. Here are some common procedures for variable selection:

Forward Selection
Forward selection is a stepwise approach where predictors are added one by one to the model based on a specified criterion (e.g., AIC, BIC). It starts with no predictors and adds variables that improve the model's performance the most.
Example:

1. Start with an empty model.
2. Evaluate the addition of each predictor using a criterion (e.g., AIC).
3. Add the predictor that most improves the model.
4. Repeat until no significant improvement is observed.

Backward Elimination
Backward elimination starts with all candidate predictors in the model and iteratively removes the least significant ones.
Example:

1. Start with a model containing all predictors.
2. Evaluate the removal of each predictor using a criterion (e.g., p-values).
3. Remove the predictor with the least impact.
4. Repeat until all remaining predictors are statistically significant.

Stepwise Selection
Stepwise selection combines forward and backward methods. It adds predictors as in forward selection and removes them as in backward elimination, aiming to find a balance.
Example:

1. Start with an empty model.
2. Add predictors based on improvement criteria (e.g., p-values).
3. After adding each predictor, remove any that no longer meet the criteria.
4. Continue until no further improvements can be made.

LASSO (Least Absolute Shrinkage and Selection Operator)
LASSO regression applies a penalty to the size of the coefficients, effectively shrinking some coefficients to zero. This results in variable selection as only the predictors with non-zero coefficients are retained.
Example:

1. Fit a LASSO model with a chosen penalty parameter (λ).
2. The penalty term encourages sparsity in the model coefficients.
3. Variables with coefficients shrunk to zero are excluded from the model.

Ridge Regression

Ridge regression also applies a penalty to the coefficients, but it does not set coefficients to zero. Instead, it shrinks all coefficients towards zero, which can be useful for multicollinearity but does not perform variable selection by itself.
Example:

1. Fit a ridge regression model with a penalty parameter (λ).
2. All predictors are included, but their coefficients are regularized.

Elastic Net
Elastic Net combines the penalties of LASSO and Ridge regression, allowing for both variable selection and coefficient shrinkage.
Example:

1. Fit an Elastic Net model with both L1 (LASSO) and L2 (Ridge) penalties.
2. The model includes a mixing parameter to balance between LASSO and Ridge effects.

Principal Component Analysis (PCA)
PCA reduces dimensionality by transforming the predictors into orthogonal components. While not a selection method per se, it can be used to select a subset of principal components that explain a significant portion of the variance.
Example:

1. Perform PCA on the predictors.
2. Select a subset of principal components that capture most of the variance.
3. Use these components as predictors in the regression model.

Best Subset Selection
Best subset selection evaluates all possible combinations of predictors and selects the one that best meets a criterion (e.g., AIC, BIC). This method is computationally intensive but can be very effective.
Example:

1. Evaluate all possible models with different subsets of predictors.
2. Choose the model that minimizes the chosen criterion.

We've looked more closely at the penalty-based models. Now we want to look at the methods that are historically done manually (the stepwise approaches, forward and backward selection), and best subset regression. Some of these methods are implemented by hand by checking statistical significance of the variables, while in R, they are sometimes implemented by checking other criteria, such as AIC and BIC. We'd like to implement them by any given criteria we'd prefer. So, let's look at how they are coded. We'll look at PCA in a later lecture.

We'll start with **best subset selection** since this one is prohibitively difficult to by hand unless you have only a handful of variables.

```
# Load necessary packages
library(MASS)  # For stepAIC if needed

# Load the mtcars dataset
data(mtcars)

# Define the response variable and predictor variables
response <- "mpg"
```

```r
predictors <- setdiff(names(mtcars), response)

# Function to calculate Adjusted R-squared
adj_r_squared <- function(model) {
  r2 <- summary(model)$r.squared
  n <- nrow(model$model)
  p <- length(model$coefficients) - 1
  adj_r2 <- 1 - (1 - r2) * (n - 1) / (n - p - 1)
  return(adj_r2)
}

# Function to fit models for each subset of predictors and evaluate
best_subset_selection <- function(data, response, predictors) {
  best_model <- NULL
  best_adj_r2 <- -Inf
  best_subset <- NULL

  # Generate all possible subsets of predictors
  for (k in 1:length(predictors)) {
    subsets <- combn(predictors, k, simplify = FALSE)

    for (subset in subsets) {
      formula <- as.formula(paste(response, "~", paste(subset, collapse = " + ")))
      model <- lm(formula, data = data)

      # Calculate Adjusted R-squared
      current_adj_r2 <- adj_r_squared(model)

      if (current_adj_r2 > best_adj_r2) {
        best_adj_r2 <- current_adj_r2
        best_model <- model
        best_subset <- subset
      }
    }
  }

  return(list(model = best_model, subset = best_subset, adj_r2 = best_adj_r2))
}

# Apply best subset selection
result <- best_subset_selection(mtcars, response, predictors)

# Print the results
cat("Best Subset:\n")
print(result$subset)
cat("\nBest Model Summary:\n")
print(summary(result$model))
cat("\nAdjusted R-squared:", result$adj_r2, "\n")
```

Let's now consider **backward selection** procedures. Normally, we eliminate variables according to their p-values until all coefficients are less than the specified threshold (usually 0.05). The code below will perform this process with some amendments: 1) it adds some additional regression metrics at the end, and 2) it does not test for elimination of the constant. This would be the last step necessary to do by hand if we wanted to.

```
# Load necessary packages
library(MASS)  # For stepAIC if needed

# Load the mtcars dataset
data(mtcars)

# Define the response variable and predictor variables
response <- "mpg"
predictors <- setdiff(names(mtcars), response)

# Function to fit the model and get summary statistics
fit_model <- function(data, response, predictors) {
  formula <- as.formula(paste(response, "~", paste(predictors, collapse = " + ")))
  model <- lm(formula, data = data)
  return(model)
}

# Function to get the highest p-value from model summary
get_highest_pvalue <- function(model) {
  summary(model)$coefficients[, "Pr(>|t|)"][-1]  # Exclude the intercept
}

# Function to perform backward selection
backward_selection <- function(data, response, predictors, threshold = 0.05) {
  current_predictors <- predictors
  repeat {
    # Fit the model with the current predictors
    model <- fit_model(data, response, current_predictors)

    # Get p-values of the predictors
    pvalues <- get_highest_pvalue(model)

    # Check if any p-value is greater than the threshold
    max_pvalue <- max(pvalues, na.rm = TRUE)
    if (max_pvalue <= threshold) {
      break
    }

    # Find the predictor with the highest p-value
    predictor_to_remove <- names(pvalues)[which.max(pvalues)]
```

```
    # Remove this predictor from the current list
    current_predictors <- setdiff(current_predictors, predictor_to_remove)
  }

  # Final model
  final_model <- fit_model(data, response, current_predictors)
  return(list(model = final_model, predictors = current_predictors))
}

# Function to calculate regression metrics
regression_metrics <- function(model) {
  residuals <- model$residuals
  fitted_values <- model$fitted.values
  n <- length(residuals)
  p <- length(model$coefficients) - 1

  # Calculate metrics
  sse <- sum(residuals^2)  # Sum of Squared Errors
  sst <- sum((mtcars[[response]] - mean(mtcars[[response]]))^2)  # Total Sum of Squares
  r_squared <- 1 - sse / sst  # R-squared
  adj_r_squared <- 1 - (1 - r_squared) * (n - 1) / (n - p - 1)  # Adjusted R-squared
  rmse <- sqrt(sse / n)  # Root Mean Squared Error
  mape <- mean(abs(residuals / mtcars[[response]])) * 100  # Mean Absolute Percentage Error
  aic <- AIC(model)  # Akaike Information Criterion
  bic <- BIC(model)  # Bayesian Information Criterion

  return(list(
    R_squared = r_squared,
    Adjusted_R_squared = adj_r_squared,
    RMSE = rmse,
    MAPE = mape,
    AIC = aic,
    BIC = bic
  ))
}

# Apply backward selection
result <- backward_selection(mtcars, response, predictors)

# Print the results
cat("Final Model Summary:\n")
print(summary(result$model))

cat("\nSelected Predictors:\n")
print(result$predictors)

cat("\nRegression Metrics:\n")
metrics <- regression_metrics(result$model)
```

```
print(metrics)
```

After backward selection, a common process is to change directions and now consider adding in non-linear terms, such as interaction terms or higher-order polynomial terms. Let's look at how this could be implemented.

```
# Load necessary libraries
library(dplyr)

# Define the initial model with variables selected through backward selection
initial_vars <- c("wt", "qsec", "am")
data <- mtcars

# Start with the initial model
model_formula <- as.formula(paste("mpg ~", paste(initial_vars, collapse = " + ")))
current_model <- lm(model_formula, data = data)

# Function to add polynomial and interaction terms
add_polynomial_and_interaction_terms <- function(data, initial_vars, current_model) {
 new_vars <- initial_vars
 max_degree <- 2

 # Consider adding polynomial terms
 for (var in initial_vars) {
  for (degree in 2:max_degree) {
   new_term <- paste0("I(", var, "^", degree, ")")
   model_formula <- as.formula(paste("mpg ~", paste(c(new_vars, new_term), collapse = " + ")))
   new_model <- lm(model_formula, data = data)

   # Check if the term is present in the coefficients
   if (new_term %in% rownames(summary(new_model)$coefficients)) {
    p_value <- summary(new_model)$coefficients[new_term, 4]

    if (p_value < 0.05) {
     new_vars <- c(new_vars, new_term)
     current_model <- new_model
    }
   }
  }
 }

 # Consider adding interaction terms
 interaction_combinations <- combn(initial_vars, 2, simplify = FALSE)
 for (interaction in interaction_combinations) {
  new_term <- paste(interaction, collapse = ":")
  model_formula <- as.formula(paste("mpg ~", paste(c(new_vars, new_term), collapse = " + ")))
  new_model <- lm(model_formula, data = data)
```

```
    # Check if the term is present in the coefficients
    if (new_term %in% rownames(summary(new_model)$coefficients)) {
      p_value <- summary(new_model)$coefficients[new_term, 4]

      if (p_value < 0.05) {
        new_vars <- c(new_vars, new_term)
        current_model <- new_model
      }
    }
  }

  return(current_model)
}

# Apply the function to add polynomial and interaction terms
final_model <- add_polynomial_and_interaction_terms(data, initial_vars, current_model)

# Output the final model summary
summary(final_model)

# Calculate additional metrics
rsq <- summary(final_model)$r.squared
adj_rsq <- summary(final_model)$adj.r.squared
aic_value <- AIC(final_model)
bic_value <- BIC(final_model)
rmse <- sqrt(mean(residuals(final_model)^2))

cat("R-squared: ", rsq, "\n")
cat("Adjusted R-squared: ", adj_rsq, "\n")
cat("AIC: ", aic_value, "\n")
cat("BIC: ", bic_value, "\n")
cat("RMSE: ", rmse, "\n")
```

In this example, we run through various quadratic and degree-2 interaction terms to see if they can be added to the model. One thing this algorithm does not do is remove variables from the initial set. So, at the end of this algorithm, it added a quadratic term, but this makes another variable in the model have a p-value that is too large, but this version of the algorithm does not check that. This is an aspect of forward selection that can be quite complex. However, this algorithm does help in that you don't have to test every possibility yourself, and can now do just a little clean-up at the end. An alternative, here, would be to add the combinations of variables to the input yourself, and run best subset selection on those options since the order terms get added in here will make a big difference.

Sometimes the issue is not the number of variables, but the different kinds of model options that are available, with each having to be compared and tested separately. Let's consider a function that will apply various linear and non-linear models to a one-variable input case, and then select the best model based on some selected regression metric.

```
# Load necessary libraries
```

```
library(mgcv)    # For GAMs and penalized splines
library(splines) # For B-splines
library(kernlab) # For Gaussian Process
#loess model is in the stats package which is already loaded in standard R

# Define a function to calculate MAPE
mape <- function(actual, predicted) {
  mean(abs((actual - predicted) / actual)) * 100
}

# Define a function to compare models
compare_models <- function(data, response, predictor) {
  # Extract response and predictor
  y <- data[[response]]
  x <- data[[predictor]]

  # Prepare data
  model_data <- data.frame(x = x, y = y)

  # Linear model
  linear_model <- lm(y ~ x, data = model_data)
  linear_pred <- predict(linear_model, newdata = model_data)
  linear_mape <- mape(y, linear_pred)

  # Quadratic model
  quadratic_model <- lm(y ~ x + I(x^2), data = model_data)
  quadratic_pred <- predict(quadratic_model, newdata = model_data)
  quadratic_mape <- mape(y, quadratic_pred)

  # Cubic model
  cubic_model <- lm(y ~ x + I(x^2) + I(x^3), data = model_data)
  cubic_pred <- predict(cubic_model, newdata = model_data)
  cubic_mape <- mape(y, cubic_pred)

  # Quartic model
  quartic_model <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = model_data)
  quartic_pred <- predict(quartic_model, newdata = model_data)
  quartic_mape <- mape(y, quartic_pred)

  # LOESS model
  loess_model <- loess(y ~ x, data = model_data)
  loess_pred <- predict(loess_model, newdata = model_data)
  loess_mape <- mape(y, loess_pred)

  # Smoothing spline model
  smooth_spline_model <- smooth.spline(x, y)
  spline_pred <- predict(smooth_spline_model, x)$y
  spline_mape <- mape(y, spline_pred)
```

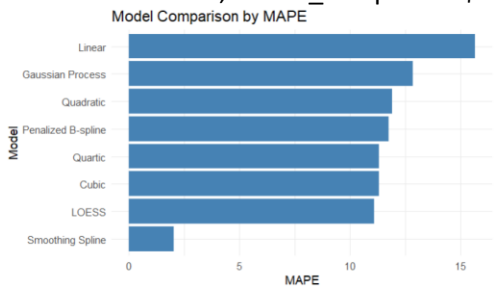
```
  # Penalized B-spline model
  penalty_spline_model <- gam(y ~ s(x, bs = "cs"), data = model_data)
  penalty_spline_pred <- predict(penalty_spline_model, newdata = model_data)
  penalty_spline_mape <- mape(y, penalty_spline_pred)

  # Gaussian Process model
  gp_model <- gausspr(x = matrix(x), y = y, kernel = rbfdot(sigma = 0.1))
  gp_pred <- predict(gp_model, matrix(x))
  gp_mape <- mape(y, gp_pred)

  # Collect results
  results <- data.frame(
    Model = c("Linear", "Quadratic", "Cubic", "Quartic", "LOESS", "Smoothing Spline", "Penalized
B-spline", "Gaussian Process"),
    MAPE = c(linear_mape, quadratic_mape, cubic_mape, quartic_mape, loess_mape,
spline_mape, penalty_spline_mape, gp_mape)
  )

  # Find the best model
  best_model <- results[which.min(results$MAPE), ]

  return(list(
    results = results,
    best_model = best_model
  ))
}

# Example usage with mtcars dataset
model_comparison <- compare_models(mtcars, "mpg", "hp")

# Print results
print(model_comparison$results)
print(paste("Best model based on MAPE:", model_comparison$best_model$Model))
```

This function prints the results as text to the output, but perhaps we'd prefer a visualization for easier comparison?

```
# Load necessary libraries
library(mgcv)    # For GAMs and penalized splines
library(splines) # For B-splines
library(kernlab) # For Gaussian Process
library(ggplot2) # For plotting

# Define a function to calculate MAPE
mape <- function(actual, predicted) {
  mean(abs((actual - predicted) / actual)) * 100
}
```

```r
# Define a function to compare models and plot results
compare_models <- function(data, response, predictor) {
 # Extract response and predictor
 y <- data[[response]]
 x <- data[[predictor]]

 # Prepare data
 model_data <- data.frame(x = x, y = y)

 # Linear model
 linear_model <- lm(y ~ x, data = model_data)
 linear_pred <- predict(linear_model, newdata = model_data)
 linear_mape <- mape(y, linear_pred)

 # Quadratic model
 quadratic_model <- lm(y ~ x + I(x^2), data = model_data)
 quadratic_pred <- predict(quadratic_model, newdata = model_data)
 quadratic_mape <- mape(y, quadratic_pred)

 # Cubic model
 cubic_model <- lm(y ~ x + I(x^2) + I(x^3), data = model_data)
 cubic_pred <- predict(cubic_model, newdata = model_data)
 cubic_mape <- mape(y, cubic_pred)

 # Quartic model
 quartic_model <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = model_data)
 quartic_pred <- predict(quartic_model, newdata = model_data)
 quartic_mape <- mape(y, quartic_pred)

 # LOESS model
 loess_model <- loess(y ~ x, data = model_data)
 loess_pred <- predict(loess_model, newdata = model_data)
 loess_mape <- mape(y, loess_pred)

 # Smoothing spline model
 smooth_spline_model <- smooth.spline(x, y)
 spline_pred <- predict(smooth_spline_model, x)$y
 spline_mape <- mape(y, spline_pred)

 # Penalized B-spline model
 penalty_spline_model <- gam(y ~ s(x, bs = "cs"), data = model_data)
 penalty_spline_pred <- predict(penalty_spline_model, newdata = model_data)
 penalty_spline_mape <- mape(y, penalty_spline_pred)

 # Gaussian Process model
 gp_model <- gausspr(x = matrix(x), y = y, kernel = rbfdot(sigma = 0.1))
 gp_pred <- predict(gp_model, matrix(x))
```

```
  gp_mape <- mape(y, gp_pred)

  # Collect results
  results <- data.frame(
    Model = c("Linear", "Quadratic", "Cubic", "Quartic", "LOESS", "Smoothing Spline", "Penalized
B-spline", "Gaussian Process"),
    MAPE = c(linear_mape, quadratic_mape, cubic_mape, quartic_mape, loess_mape,
spline_mape, penalty_spline_mape, gp_mape)
  )

  # Find the best model
  best_model <- results[which.min(results$MAPE), ]

  # Plot results
  p <- ggplot(results, aes(x = reorder(Model, MAPE), y = MAPE)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    coord_flip() +
    labs(title = "Model Comparison by MAPE", x = "Model", y = "MAPE") +
    theme_minimal()

  # Print plot
  print(p)

  return(list(
    results = results,
    best_model = best_model
  ))
}

# Example usage with mtcars dataset
model_comparison <- compare_models(mtcars, "mpg", "hp")

# Print results
print(model_comparison$results)
print(paste("Best model based on MAPE:", model_comparison$best_model$Model))
```



Selection algorithms of this sort can follow traditional processes or they can be customized to help facilitate model selection processes. You won't necessarily be able to automate every step of the process, as we've seen, but they can go a long way toward making valuable assessments of initial options so that we as data analysts can focus on the final steps of the selection process.

Resources:

1. https://www.biostat.jhsph.edu/~iruczins/teaching/jf/ch10.pdf