Supervised Learning

James Caldwell

2024-02-26

Feb 2024

Overview

This document outlines a supervised learning process involving the generation of synthetic data, fitting polynomial regression models, and evaluating their performance using mean squared error (MSE). The script explores the effects of varying parameters such as sample size and standard deviation on model fitting and prediction accuracy.

Script Contents

The script consists of the following sections:

1. **Data Generation**: Functions to simulate x and y values from normal distributions and define the true mean function.
2. **Data Visualization**: Scatterplots of the training data along with the true regression line.
3. **Model Fitting**: Polynomial regression models (linear, quadratic, cubic) fitted to the training data.
4. **Model Evaluation**: Prediction of test data using the trained models and calculation of MSE for each model.
5. **Simulation and Analysis**: Simulation of multiple evaluations with varying parameters to explore model performance under different conditions.

Observations

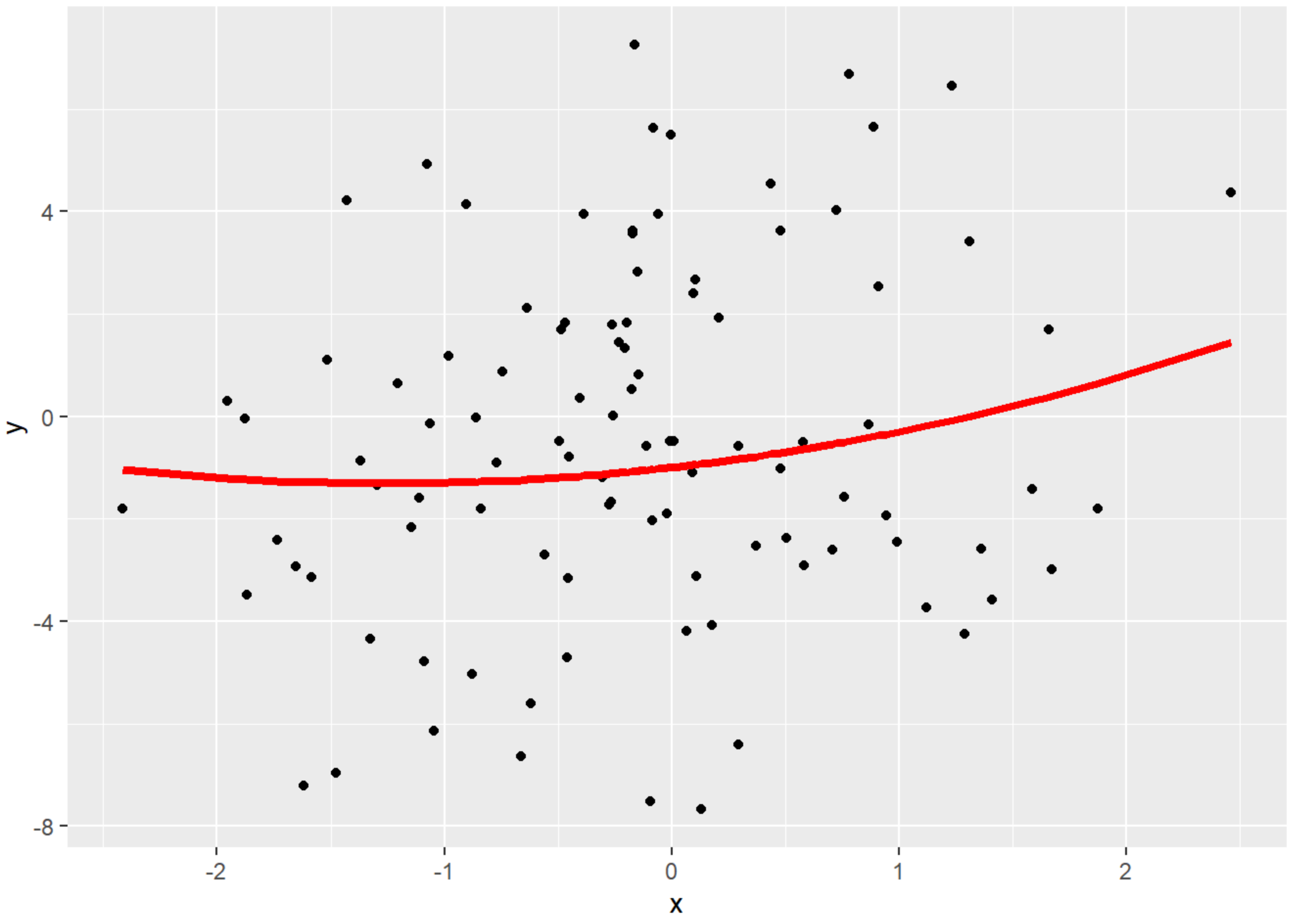
- Increasing the standard deviation (sigma) introduces more noise, making the data less representative of the true model form.
- Larger sample sizes (n) provide a clearer picture of the data trends, leading to more accurate model fitting.
- The best model for prediction may not always be the true model form, especially when the data is highly variable.

For further details and code implementation, refer to the R script.

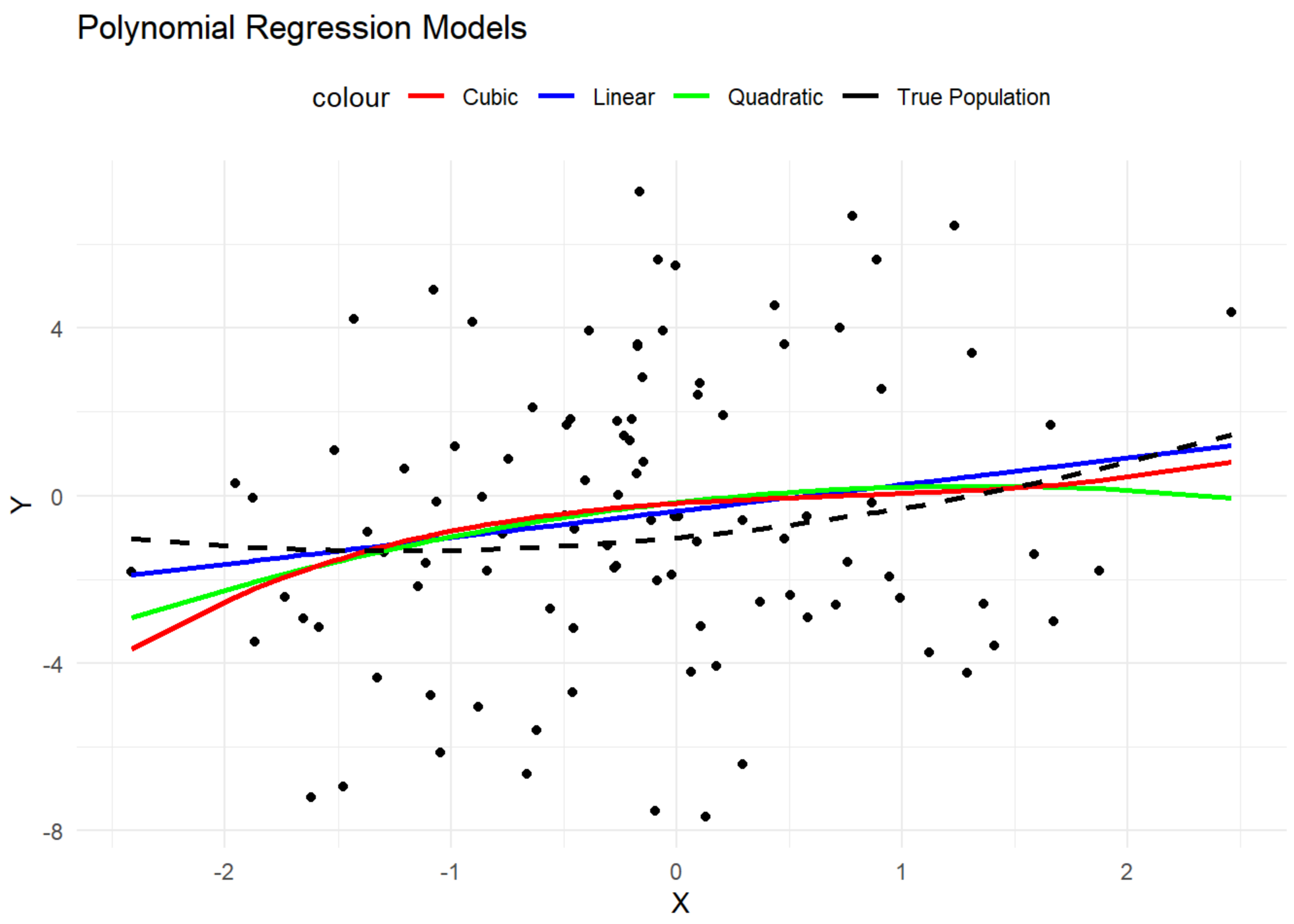
```r
# James Caldwell
# Feb 2024

library(tidyverse) # functions for data manipulation
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✓ forcats   1.0.0     ✓ stringr   1.5.1
## ✓ ggplot2   3.4.4     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.0
## ✓ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✕ dplyr::filter() masks stats::filter()
## ✕ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(ggplot2)

# Function to simulate x values from a normal distribution
sim_x <- function(n) rnorm(n)

# True mean function for generating y values
f <- function(x) -1 + 0.5 * x + 0.2 * x^2

# Function to simulate y values with added noise
sim_y <- function(x, sd) {
  f(x) + rnorm(length(x), sd = sd)
}

# Set seed for reproducibility
set.seed(611)

# Number of observations
n = 100

# Standard deviation
sd = 3

# Simulate x and y values
x = sim_x(n)
y = sim_y(x, sd = sd)

# True regression function
true_regression <- function(x) -1 + 0.5*x + 0.2*(x)^2

# Create tibble for training data
data_train <- tibble(x, y)

# Scatterplot with true regression line
ggplot(data_train, aes(x, y)) +
  geom_point() +
  geom_line(aes(y = true_regression(x)), color = "red", linewidth = 1.5)
```



```
# Fit polynomial regression models
linear_model <- lm(y ~ x, data = data_train)
quadratic_model <- lm(y ~ poly(x, degree = 2), data = data_train)
cubic_model <- lm(y ~ poly(x, degree = 3), data = data_train)

# Predictions
data_train$linear_pred <- predict(linear_model)
data_train$quadratic_pred <- predict(quadratic_model)
data_train$cubic_pred <- predict(cubic_model)

# Plot regression models
ggplot(data_train, aes(x, y)) +
  geom_point() +
  geom_line(aes(x, linear_pred, color = "Linear"), linewidth = 1) +
  geom_line(aes(x, quadratic_pred, color = "Quadratic"), linewidth = 1) +
  geom_line(aes(x, cubic_pred, color = "Cubic"), linewidth = 1) +
  geom_line(aes(x, -1 + 0.5 * x + 0.2 * x^2, color = "True Population"), linewidth = 1, linetype = "dashe
d") +
  scale_color_manual(values = c("Linear" = "blue", "Quadratic" = "green", "Cubic" = "red", "True Populatio
n" = "black")) +
  labs(title = "Polynomial Regression Models",
       x = "X",
       y = "Y") +
  theme_minimal() +
  theme(legend.position = "top")
```



```
# Set seed for reproducibility
set.seed(612)

# Number of test observations
n_test = 10000

# Standard deviation for test data
sd = 3

# Simulate test data
x = sim_x(n_test)
y = sim_y(x, sd = sd)
data_test = data.frame(x, y)

# Predictions on test data and calculate MSE
data_test$linear_pred <- predict(linear_model, newdata = data.frame(x = data_test$x_test))
data_test$quadratic_pred <- predict(quadratic_model, newdata = data.frame(x = data_test$x_test))
data_test$cubic_pred <- predict(cubic_model, newdata = data.frame(x = data_test$x_test))

mse_linear <- mean((data_test$linear_pred - data_test$y)^2)
mse_quadratic <- mean((data_test$quadratic_pred - data_test$y)^2)
mse_cubic <- mean((data_test$cubic_pred - data_test$y)^2)

cat("MSE for Linear Model:", mse_linear, "\n")
```

```
## MSE for Linear Model: 9.293776
```

```
cat("MSE for Quadratic Model:", mse_quadratic, "\n")
```

```
## MSE for Quadratic Model: 9.583155
```

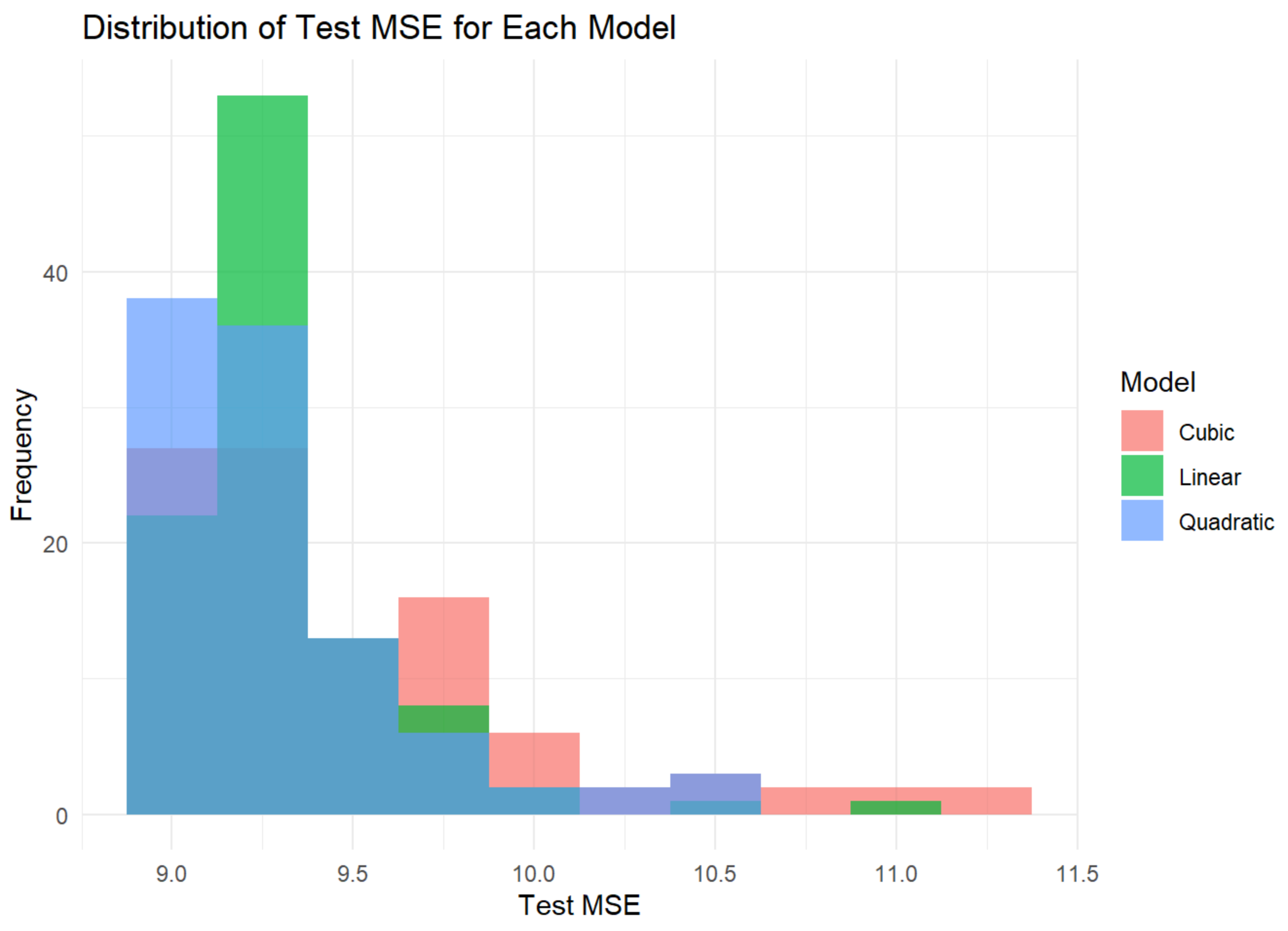
```
cat("MSE for Cubic Model:", mse_cubic, "\n")
```

```
## MSE for Cubic Model: 9.648192
```

```
# Compute the mean squared error of the baseline model by calculating the variance of y.
mse_best <- var(y)

# Print the computed mean squared error of the baseline model.
print(mse_best)
```

```
## [1] 9.311119
```

```r
# Define a function to simulate training data, fit regression models, make predictions on test data,
# and calculate the mean squared error for each model.
simulate_and_evaluate <- function(data_test, n_train, sd_train) {

  # Simulate training data
  x <- sim_x(n_train)
  y <- sim_y(x, sd = sd_train)
  data_train <- tibble(x, y)

  # Fit regression models (linear, quadratic, and cubic)
  linear_model <- lm(y ~ x, data = data_train)
  quadratic_model <- lm(y ~ poly(x, 2), data = data_train)
  cubic_model <- lm(y ~ poly(x, 3), data = data_train)

  # Make predictions on test data
  data_test$linear_pred <- suppressWarnings({predict(linear_model, newdata = data_test)})
  data_test$quadratic_pred <- suppressWarnings({predict(quadratic_model, newdata = data_test)})
  data_test$cubic_pred <- suppressWarnings({predict(cubic_model, newdata = data_test)})

  # Calculate mean squared error for each model
  mse_linear <- mean((data_test$linear_pred - data_test$y)^2)
  mse_quadratic <- mean((data_test$quadratic_pred - data_test$y)^2)
  mse_cubic <- mean((data_test$cubic_pred - data_test$y)^2)

  # Return the mean squared errors for linear, quadratic, and cubic models
  return(c(mse_linear, mse_quadratic, mse_cubic))
}

# Run simulations to evaluate the performance of linear, quadratic, and cubic models
set.seed(613)
n_train <- 100
sd_train <- 3
num_simulations <- 100
results <- replicate(num_simulations, simulate_and_evaluate(data_test, n_train, sd_train))

# Convert simulation results to a data frame
results_df <- data.frame(t(results))

# Set column names for the data frame
colnames(results_df) <- c("Linear", "Quadratic", "Cubic")

# Reshape data into long format for visualization
gathered_results <- gather(results_df, key = "Model", value = "MSE", Linear, Quadratic, Cubic)

# Create histogram plots to visualize the distribution of test mean squared errors for each model
ggplot(gathered_results, aes(x = MSE, fill = Model)) +
  geom_histogram(binwidth = 0.25, position = "identity", alpha = 0.7) +
  labs(title = "Distribution of Test MSE for Each Model",
       x = "Test MSE",
       y = "Frequency") +
  theme_minimal()
```



```r
# Initialize counts to track the best model in each simulation
best_model_counts <- c(Linear = 0, Quadratic = 0, Cubic = 0)

# Loop through each simulation
for (i in 1:num_simulations) {
  # Identify the model with the lowest mean squared error in each simulation
  best_model <- names(results_df)[which.min(results_df[i, ])]
  # Increment the count for the best model
  best_model_counts[best_model] <- best_model_counts[best_model] + 1
}

# Create a data frame to display the counts of the best model in each simulation
counts_df <- data.frame(Model = names(best_model_counts), Count = best_model_counts)

# Print the counts of the best model in each simulation
print(counts_df)
```

```
##               Model Count
## Linear       Linear    28
## Quadratic Quadratic    65
## Cubic         Cubic     7
```

Increasing σ increases the spread within the original data. Thus, increasing σ then makes the data look less and less like the original function used to generate the data. This allows for other model forms (ex: a linear model) to more accurately describe the data than the true model form (ex: a quadratic).

n has a less pronounced effect than σ does, however increasing n generally allows for a more accurate “picture” of the data for a model to use for interpreting trends. Thus, increasing n should lead towards a model that is closer to the true data form. However, after a certain point, the data shows enough of a picture and increasing n probably does nothing more than add more noise to the data.

The true model form is not always the goal for a prediction because we want to model the actual data where it actually lies. Large variation (large σ), zoomed in range of data, and other factors can distort the data enough that there may be a model that describes the data better than the true model form.