Statistical Learning - Assignment 01

Joshua Damm

2024-02-14

Write a function that…

```r
set.seed(42)

# Function to generate data
generate_data <- function(n) {
  x <- runif(n, -3, 3)
  epsilon <- rnorm(n, 0, 1)
  y <- 8 * sin(x) + epsilon
  train <- data.frame(x = x, y = y)
  return(train)
}
```

Use this function to generate a training set of size 50 and a test set of size 10000…

```r
# set train and test sizes
train_size_train = 50
test_size_train = 10000


# Generate test data
train_data = generate_data(train_size_train)
test_data <- generate_data(test_size_train)
```

Train two polynomial regression models…

```r
# train two poly regression models (3, and 15)

#3 degrees
fit_3 <- lm(y ~ poly(x, degree = 3), data = train_data)
train_pred <- predict(fit_3, newdata = train_data)
test_pred <- predict(fit_3, newdata = test_data)
test_err_3 <- mean((test_data$y - test_pred)^2)
```

```r
train_err <- mean((train_data$y - train_pred)^2)

#15 degrees
fit_15 <- lm(y ~ poly(x, degree = 15), data = train_data)
train_pred <- predict(fit_15, newdata = train_data)
test_pred <- predict(fit_15, newdata = test_data)
test_err_15 <- mean((test_data$y - test_pred)^2)
train_err <- mean((train_data$y - train_pred)^2)

# get MSE on test set
test_err_3; test_err_15;
```

```
## [1] 1.258852
```

```
## [1] 1.632571
```

```r
# repeat with training set of size 10000
train_size_train = 10000

# Generate test data
train_data = generate_data(train_size_train)
test_data <- generate_data(test_size_train)


# train two poly regression models (3, and 15)

#3 degrees
fit_3 <- lm(y ~ poly(x, degree = 3), data = train_data)
train_pred <- predict(fit_3, newdata = train_data)
test_pred <- predict(fit_3, newdata = test_data)
test_err_3.1 <- mean((test_data$y - test_pred)^2)
train_err <- mean((train_data$y - train_pred)^2)

#15 degrees
fit_15 <- lm(y ~ poly(x, degree = 15), data = train_data)
train_pred <- predict(fit_15, newdata = train_data)
test_pred <- predict(fit_15, newdata = test_data)
test_err_15.1 <- mean((test_data$y - test_pred)^2)
train_err <- mean((train_data$y - train_pred)^2)

# get MSE on test set
test_err_3.1; test_err_15.1;
```

```
## [1] 1.205183
```

```
## [1] 1.00663
```

Prediction rule f / Obtain test MSE for best prediction rule...

```
test_err_15.1
```

```
## [1] 1.00663
```

The optimal prediction rule is determined by selecting the model that yields the lowest Mean Squared Error (MSE) on the training data. In the context of 10,000 training data points, the polynomial model of degree 15 demonstrates the lowest MSE, specifically measured at "1.00663," as indicated earlier. Therefore, for this dataset, the polynomial of degree 15 is considered the most effective predictive model based on the training data.

Report 4 MSE's, explained with bias / variance…

```
test_err_3; test_err_15;test_err_3.1; test_err_15.1
```

```
## [1] 1.258852
```

```
## [1] 1.632571
```

```
## [1] 1.205183
```

```
## [1] 1.00663
```

Achieving a lower Mean Squared Error (MSE) is indicative of a superior predictor, with MSE being influenced by both Bias and Variance. Ideally, one would aim for models with low Bias and Variance, but in practice, there exists a trade-off between the two. An effective predictor should strike a balance, being flexible enough to capture underlying relationships in the data (low bias) while avoiding excessive complexity that may lead to poor performance on new data (high variance).

In the context of a substantial training set, a higher-degree polynomial outperforms others as it accommodates the intricate structure within the data. However, for a smaller training set, a 3-degree polynomial proves to be more effective. Its lower flexibility is sufficient to capture the data's structure, making it perform better on new data compared to a higher-degree polynomial, which might overfit the smaller dataset.