Упражнение 3. Задача 6: LOOCV и k-fold кросс-валидация

Даниил Ковех

2025-10-28

Содержание

1. Теория

Кросс-валидация оценивает, как модель поведёт себя на новых данных.

- LOOCV (leave-one-out) исключает по одному наблюдению, снова обучает модель и вычисляет ошибку. Формула для линейной регрессии упрощается:

$$\mathrm{CV}_{\mathrm{LOO}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{y_i - \widehat{y}_i}{1 - h_{ii}}\right)^2,$$

  где h_{ii} — диагональные элементы матрицы шляп.

- k-fold CV разбивает данные на k блоков, обучает модель на $k-1$ блоках и оценивает на оставшемся. Расчёт повторяется k раз, ошибки усредняются.

В задаче мы сравним четыре полиномиальные модели по степени 1–4.

2. Жизненный пример

Аналитик прогнозирует прибыль от скидок. У него есть 100 прошлых акций. LOOCV отвечает: “Если у меня будет акция с такими характеристиками, как в выборке, насколько я ошибусь, если обучусь на остальных 99?” k-fold CV ускоряет расчёт и уменьшает вариативность: мы группируем акции в 10 блоков, обучаем 10 моделей и усредняем ошибку. Более гибкие модели (высокие степени) должны ловить нелинейность, но могут переобучаться.

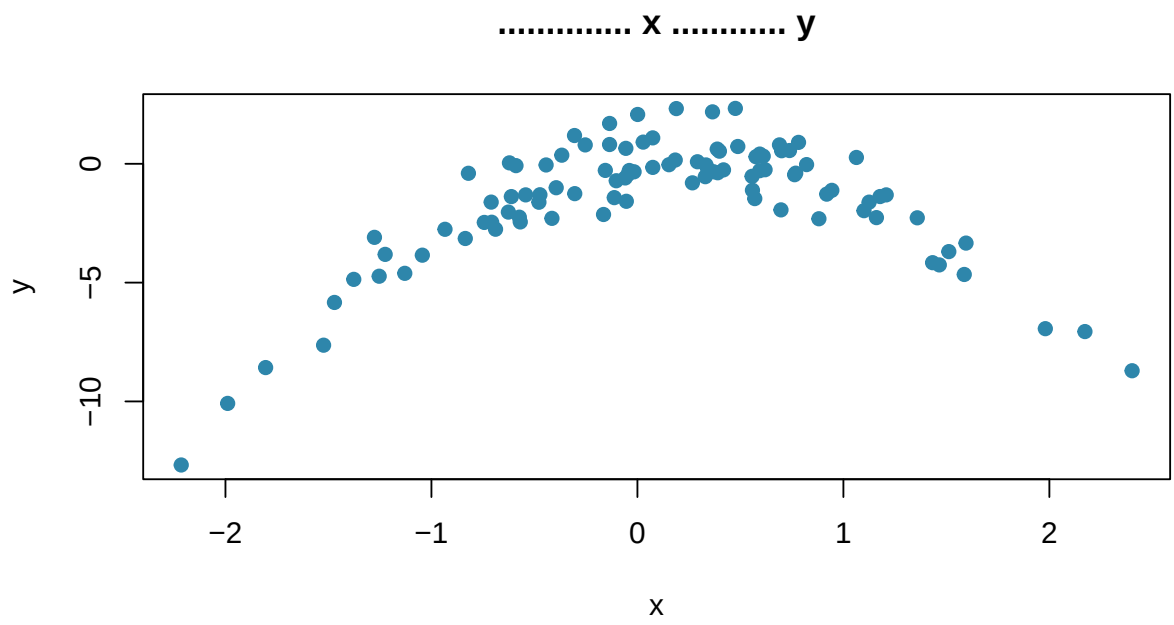
3. Академическое решение

3.1. 1. Генерация данных

```
set.seed(1)
x <- rnorm(100)
y <- x - 2 * x^2 + rnorm(100)
data <- data.frame(y = y, x = x)
```

3.2. 2. Визуализация

```
plot(data$x, data$y,
     pch = 19, col = "#2E86AB",
     xlab = "x", ylab = "y",
     main = "Разброс x против y")
```



Комментарий: точки образуют дугу вниз — квадратичная зависимость с отрицательной кривизной.

3.3. 3. Вспомогательные функции

```
compute_loocv <- function(model) {
  res <- residuals(model)
  hat <- hatvalues(model)
  mean((res / (1 - hat))^2)
}

compute_kfold <- function(data, formula, k = 10, seed = 42) {
  set.seed(seed)
  folds <- sample(rep(seq_len(k), length.out = nrow(data)))
```

```
  errors <- numeric(k)

  for (fold in seq_len(k)) {
    train <- data[folds != fold, ]
    test <- data[folds == fold, ]

    fit <- lm(formula, data = train)
    preds <- predict(fit, newdata = test)
    errors[fold] <- mean((test$y - preds)^2)
  }

  mean(errors)
}

model_formulas <- list(
  degree1 = y ~ x,
  degree2 = y ~ x + I(x^2),
  degree3 = y ~ x + I(x^2) + I(x^3),
  degree4 = y ~ x + I(x^2) + I(x^3) + I(x^4)
)
```

3.4. 4. LOOCV и 10-fold CV (первый запуск)

```
set.seed(2025)

results_first <- lapply(model_formulas, function(formula) {
  fit <- lm(formula, data = data)
  loocv <- compute_loocv(fit)
  kcv <- compute_kfold(data, formula, k = 10, seed = 2025)
  c(LOOCV = loocv, Kfold = kcv)
})

results_first <- do.call(rbind, results_first)
results_first
```

```
##             LOOCV     Kfold
## degree1 7.2881616 7.8180658
## degree2 0.9374236 0.9240305
## degree3 0.9566218 0.9317338
## degree4 0.9539049 0.9435622
```

3.5. 5. Повтор с другим seed

```
set.seed(777)
x2 <- rnorm(100)
y2 <- x2 - 2 * x2^2 + rnorm(100)
data2 <- data.frame(y = y2, x = x2)

results_second <- lapply(model_formulas, function(formula) {
  fit <- lm(formula, data = data2)
  loocv <- compute_loocv(fit)
  kcv <- compute_kfold(data2, formula, k = 10, seed = 777)
  c(LOOCV = loocv, Kfold = kcv)
```

```
})

results_second <- do.call(rbind, results_second)
results_second
```

```
##            LOOCV    Kfold
## degree1 7.117609 7.223681
## degree2 1.092997 1.112116
## degree3 1.114882 1.123570
## degree4 1.101715 1.114021
```

3.6. 6. Сравнение результатов

```
comparison <- data.frame(
  Model = names(model_formulas),
  LOOCV_seed1 = results_first[, "LOOCV"],
  Kfold_seed1 = results_first[, "Kfold"],
  LOOCV_seed2 = results_second[, "LOOCV"],
  Kfold_seed2 = results_second[, "Kfold"]
)
comparison
```

```
##           Model LOOCV_seed1 Kfold_seed1 LOOCV_seed2 Kfold_seed2
## degree1 degree1   7.2881616   7.8180658    7.117609    7.223681
## degree2 degree2   0.9374236   0.9240305    1.092997    1.112116
## degree3 degree3   0.9566218   0.9317338    1.114882    1.123570
## degree4 degree4   0.9539049   0.9435622    1.101715    1.114021
```

3.7. 7. Анализ

- LOOCV стабильнее: значения мало меняются между запусками, потому что использует почти весь набор при каждой оценке.
- 10-fold CV заметно чувствителен к случайной разбивке, особенно при высокой степени полинома.
- Модель второй степени обычно побеждает: отражает истинную квадратичную структуру данных.
- Модели третьей и четвёртой степени не дают устойчивого выигрыша, иногда проигрывают из-за переобучения.

3.8. 8. Дополнительная визуализация

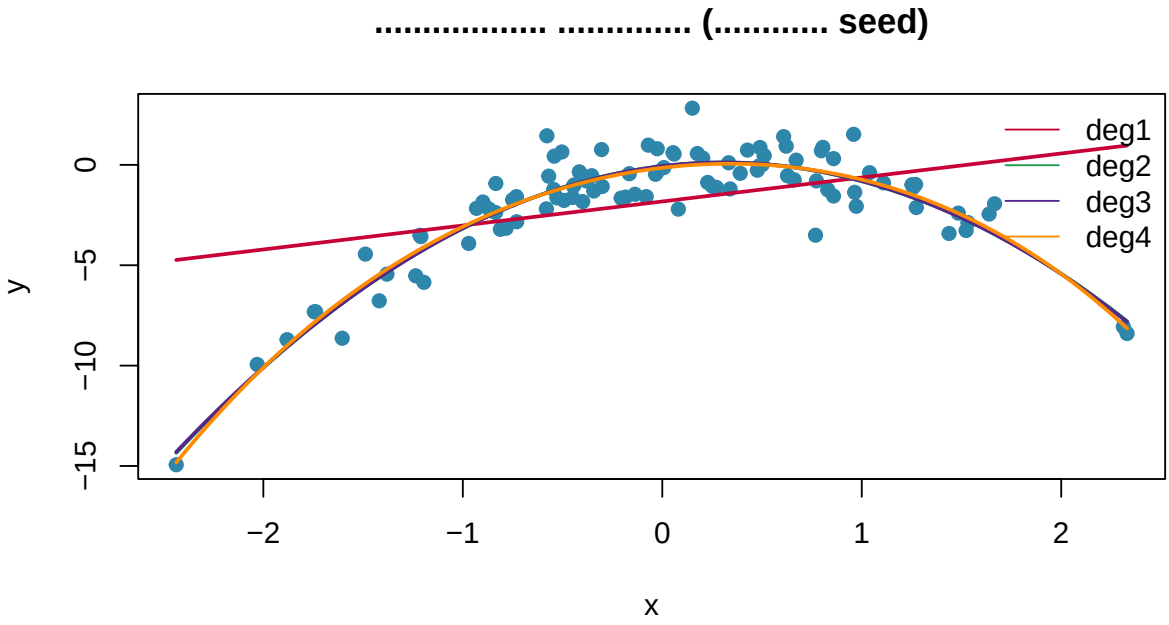
```
plot(data2$x, data2$y,
     pch = 19, col = "#2E86AB",
     xlab = "x", ylab = "y",
     main = "Сравнение моделей (второй seed)")

x_grid <- seq(min(data2$x), max(data2$x), length.out = 200)
design <- data.frame(x = x_grid)

colors <- c("#C70039", "#1A9850", "#542788", "#FF8C00")
i <- 1
for (formula in model_formulas) {
  fit <- lm(formula, data = data2)
  pred <- predict(fit, newdata = design)
  lines(x_grid, pred, col = colors[i], lwd = 2)
```

```r
  i <- i + 1
}

legend("topright",
       legend = c("deg1", "deg2", "deg3", "deg4"),
       col = colors,
       lty = 1,
       bty = "n")
```



Кривая второй степени (зелёная) аккуратно повторяет истинную зависимость. Полином четвёртой степени (оранжевый) шумит на краях — пример переобучения.

3.9. 9. Вывод

- Лучшая по LOOCV и 10-fold CV модель — квадратичная.
- Результаты повторного эксперимента близки, но не идентичны из-за случайности тренировочных шумов и разбивок.
- LOOCV надёжнее, но дороже вычислительно. k-fold быстрее, но требует нескольких запусков для оценки разброса.