Упражнение 2: Регуляризованные обобщенные линейные модели (Задачи 1-6)

Даниил Ковех

2025-10-21

Содержание

1. Введение

Это упражнение посвящено регуляризованным обобщенным линейным моделям. Мы разберем шесть задач пошагово, объясняя каждую концепцию простым языком с реальными примерами.

2. Задача 1: Ridge регрессия и центрирование данных

2.1. Часть 1: Связь между β^{ridge} и β^{c}

2.1.1. Объяснение простыми словами

Представьте, что вы изучаете зависимость цены квартиры от её площади. Ridge регрессия - это как добавить “штраф” за слишком большие коэффициенты, чтобы модель не переобучалась.

Математическая формулировка:

Ridge регрессия с исходными данными:

$$\hat{\beta}^{ridge} = \arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Ridge регрессия с центрированными данными:

$$\hat{\beta}^{c} = \arg\min_{\beta^c} \frac{1}{N} \sum_{i=1}^{N} (y_i - \bar{y} - \beta_0^c - \sum_{j=1}^{p} (x_{ij} - \bar{x}_j)\beta_j^c)^2 + \lambda \sum_{j=1}^{p} (\beta_j^c)^2$$

где $\bar{x}_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$ и $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$

2.1.2. Связь между коэффициентами

```r
# Демонстрация связи между ridge регрессией с центрированными и нецентрированными данными

# Создаем искусственные данные
set.seed(123)
n <- 100
p <- 3
X <- matrix(rnorm(n * p), nrow = n, ncol = p)
y <- X %*% c(2, -1, 0.5) + rnorm(n, 0, 0.5)

# Вычисляем средние
x_bar <- colMeans(X)
y_bar <- mean(y)

# Центрируем данные
X_centered <- scale(X, center = TRUE, scale = FALSE)
y_centered <- y - y_bar

# Ridge регрессия с lambda = 0.1
lambda <- 0.1

# Функция для ridge регрессии
ridge_regression <- function(X, y, lambda) {
  n <- nrow(X)
  p <- ncol(X)

  # Добавляем столбец единиц для intercept
  X_with_intercept <- cbind(1, X)

  # Ridge решение
```

```
  beta_ridge <- solve(t(X_with_intercept) %*% X_with_intercept + lambda * diag(p + 1)) %*% t(X_with_inte

  return(beta_ridge)
}

# Ridge регрессия на исходных данных
beta_ridge <- ridge_regression(X, y, lambda)

# Ridge регрессия на центрированных данных
beta_centered <- ridge_regression(X_centered, y_centered, lambda)

cat("Ridge коэффициенты на исходных данных:\n")
```

```
## Ridge коэффициенты на исходных данных:
```

```
print(beta_ridge)
```

```
##              [,1]
## [1,] -0.009224089
## [2,]  1.969820485
## [3,] -0.975924440
## [4,]  0.470443849
```

```
cat("\nRidge коэффициенты на центрированных данных:\n")
```

```
##
## Ridge коэффициенты на центрированных данных:
```

```
print(beta_centered)
```

```
##               [,1]
## [1,]  2.602085e-17
## [2,]  1.969822e+00
## [3,] -9.759255e-01
## [4,]  4.704453e-01
```

```
# Связь между коэффициентами
cat("\nСвязь между коэффициентами:\n")
```

```
##
## Связь между коэффициентами:
```

```
cat("β₀^ridge =", beta_ridge[1], "\n")
```

```
## β₀^ridge = -0.009224089
```

```
cat("β₀^c =", beta_centered[1], "\n")
```

```
## β₀^c = 2.602085e-17
```

```
cat("β₀^ridge - β₀^c =", beta_ridge[1] - beta_centered[1], "\n")
```

```
## β₀^ridge - β₀^c = -0.009224089
```

```
cat("ȳ - Σ(x̄□ * β□^c) =", y_bar - sum(x_bar * beta_centered[2:4]), "\n")
```

```
## ȳ - Σ(x̄□ * β□^c) = -0.009233702
```

Вывод: $\hat{\beta}_0^{ridge} = \hat{\beta}_0^c + \bar{y} - \sum_{j=1}^{p} \bar{x}_j \hat{\beta}_j^c$

Для остальных коэффициентов: $\hat{\beta}_j^{ridge} = \hat{\beta}_j^c$ для $j = 1, \dots, p$

2.2. Часть 2: Байесовская интерпретация Ridge регрессии

2.2.1. Объяснение простыми словами

Ridge регрессия имеет красивую байесовскую интерпретацию. Это как сказать: “Я верю, что все коэффициенты должны быть маленькими, но не знаю точно насколько маленькими.”

Математическая формулировка:

Априорное распределение: $\beta \sim N(0, \tau^2 I)$ Модель выборки: $y \sim N(X\beta, \sigma^2 I)$

Апостериорное распределение: $\beta|y \sim N(\hat{\beta}^{ridge}, \Sigma)$

где $\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T y$

2.2.2. Связь между параметрами

```
# Демонстрация байесовской интерпретации
library(MASS)

# Параметры
tau_sq <- 1  # Дисперсия априорного распределения
sigma_sq <- 0.5  # Дисперсия ошибки
lambda <- sigma_sq / tau_sq  # Параметр регуляризации

cat("Параметры:\n")
```

```
## Параметры:
```

```
cat("τ² =", tau_sq, "\n")
```

```
## τ² = 1
```

```
cat("σ² =", sigma_sq, "\n")
```

```
## σ² = 0.5
```

```
cat("λ = σ²/τ² =", lambda, "\n")
```

```
## λ = σ²/τ² = 0.5
```

```
# Проверяем связь
cat("\nСвязь: λ = σ²/τ² =", sigma_sq, "/", tau_sq, "=", lambda, "\n")
```

```
##
## Связь: λ = σ²/τ² = 0.5 / 1 = 0.5
```

Вывод: $\lambda = \frac{\sigma^2}{\tau^2}$

3. Задача 2: Ортогональная матрица дизайна

3.1. Объяснение простыми словами

Представьте, что у вас есть данные, где все переменные независимы друг от друга (как рост и вес у людей - они связаны, но не линейно зависимы). В этом случае формулы для разных методов регуляризации становятся очень простыми.

Условие: $\frac{1}{N} X^T X = I$ (ортогональная матрица)

3.1.1. (a) Best subset of size M

```
# Демонстрация best subset selection
set.seed(123)
n <- 100
p <- 5

# Создаем ортогональную матрицу
X <- matrix(rnorm(n * p), nrow = n)
X <- scale(X)  # Стандартизируем
X <- X / sqrt(n)  # Делаем ортогональной

# Проверяем ортогональность
cat("Проверка ортогональности (должно быть близко к единичной матрице):\n")
```

```
## Проверка ортогональности (должно быть близко к единичной матрице):
```

```
print(round(t(X) %*% X, 3))
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]
## [1,]  0.990 -0.049 -0.128 -0.044 -0.191
## [2,] -0.049  0.990  0.030  0.043 -0.129
## [3,] -0.128  0.030  0.990 -0.044 -0.025
## [4,] -0.044  0.043 -0.044  0.990 -0.019
## [5,] -0.191 -0.129 -0.025 -0.019  0.990
```

```
# Генерируем y
y <- X %*% c(3, -2, 1, 0, 0) + rnorm(n, 0, 0.1)

# OLS оценки
beta_ols <- t(X) %*% y
cat("\nOLS коэффициенты:\n")
```

```
##
## OLS коэффициенты:
```

```
print(beta_ols)
```

```
##            [,1]
## [1,]  2.8876962
## [2,] -1.9904389
## [3,]  0.5627241
## [4,] -0.3456801
## [5,] -0.1463016
```

```
# Best subset для M = 3
M <- 3
ranks <- rank(-abs(beta_ols))  # Ранги по убыванию абсолютных значений
beta_subset <- beta_ols * (ranks <= M)

cat("\nBest subset коэффициенты (M =", M, "):\n")
```

```
##
## Best subset коэффициенты (M = 3 ):
```

```
print(beta_subset)
```

```
##            [,1]
```

```
## [1,]  2.8876962
## [2,] -1.9904389
## [3,]  0.5627241
## [4,]  0.0000000
## [5,]  0.0000000
```

Формула: $\hat{\beta}_j^{subset} = \hat{\beta}_j^{OLS} \cdot I(\text{rank}(|\hat{\beta}_j^{OLS}|) \leq M)$

3.1.2. (b) Ridge регрессия

```r
# Ridge регрессия для ортогональной матрицы
lambda <- 0.5
beta_ridge <- beta_ols / (1 + lambda)

cat("Ridge коэффициенты (λ =", lambda, "):\n")
```

```
## Ridge коэффициенты (λ = 0.5 ):
```

```r
print(beta_ridge)
```

```
##             [,1]
## [1,]  1.92513077
## [2,] -1.32695929
## [3,]  0.37514943
## [4,] -0.23045337
## [5,] -0.09753443
```

```r
cat("\nПроверка формулы: β_ridge = β_ols / (1 + λ)\n")
```

```
##
## Проверка формулы: β_ridge = β_ols / (1 + λ)
```

```r
cat("β_ols[1] / (1 + λ) =", beta_ols[1], "/ (1 +", lambda, ") =", beta_ols[1] / (1 + lambda), "\n")
```

```
## β_ols[1] / (1 + λ) = 2.887696 / (1 + 0.5 ) = 1.925131
```

```r
cat("β_ridge[1] =", beta_ridge[1], "\n")
```

```
## β_ridge[1] = 1.925131
```

Формула: $\hat{\beta}_j^{ridge} = \frac{\hat{\beta}_j^{OLS}}{1+\lambda}$

3.1.3. (c) Lasso регрессия

```r
# Lasso регрессия для ортогональной матрицы
lambda <- 0.5

# Функция soft thresholding
soft_threshold <- function(x, lambda) {
  sign(x) * pmax(abs(x) - lambda, 0)
}

beta_lasso <- soft_threshold(beta_ols, lambda)

cat("Lasso коэффициенты (λ =", lambda, "):\n")
```

```
## Lasso коэффициенты (λ = 0.5 ):
```

```r
print(beta_lasso)
```

```
##             [,1]
## [1,]  2.38769615
## [2,] -1.49043893
## [3,]  0.06272414
## [4,]  0.00000000
## [5,]  0.00000000
```

```r
cat("\nПроверка формулы: β_lasso = sign(β_ols) * (|β_ols| - λ)_+\n")
```

```
##
## Проверка формулы: β_lasso = sign(β_ols) * (|β_ols| - λ)_+
```

```r
for(i in 1:length(beta_ols)) {
  cat("β_ols[", i, "] =", beta_ols[i], "\n")
  cat("sign =", sign(beta_ols[i]), ", |β_ols| =", abs(beta_ols[i]), "\n")
  cat("(|β_ols| - λ)_+ = max(", abs(beta_ols[i]), "-", lambda, ", 0) =", pmax(abs(beta_ols[i]) - lambda,
  cat("β_lasso[", i, "] =", beta_lasso[i], "\n\n")
}
```

```
## β_ols[ 1 ] = 2.887696
## sign = 1 , |β_ols| = 2.887696
## (|β_ols| - λ)_+ = max( 2.887696 - 0.5 , 0) = 2.387696
## β_lasso[ 1 ] = 2.387696
##
## β_ols[ 2 ] = -1.990439
## sign = -1 , |β_ols| = 1.990439
## (|β_ols| - λ)_+ = max( 1.990439 - 0.5 , 0) = 1.490439
## β_lasso[ 2 ] = -1.490439
##
## β_ols[ 3 ] = 0.5627241
## sign = 1 , |β_ols| = 0.5627241
## (|β_ols| - λ)_+ = max( 0.5627241 - 0.5 , 0) = 0.06272414
## β_lasso[ 3 ] = 0.06272414
##
## β_ols[ 4 ] = -0.3456801
## sign = -1 , |β_ols| = 0.3456801
## (|β_ols| - λ)_+ = max( 0.3456801 - 0.5 , 0) = 0
## β_lasso[ 4 ] = 0
##
## β_ols[ 5 ] = -0.1463016
## sign = -1 , |β_ols| = 0.1463016
## (|β_ols| - λ)_+ = max( 0.1463016 - 0.5 , 0) = 0
## β_lasso[ 5 ] = 0
```

Формула: $\hat{\beta}_j^{lasso} = \text{sign}(\hat{\beta}_j^{OLS}) \cdot (|\hat{\beta}_j^{OLS}| - \lambda)_+$

4. Задача 3: Пуассоновская регрессия с лог-связью

4.1. Объяснение простыми словами

Представьте, что вы изучаете количество звонков в call-центр в зависимости от дня недели. Пуассоновская регрессия идеально подходит для подсчета событий.

Условия задачи: - $y_i \sim \text{Poisson}(\mu_i)$ - $g(\mu_i) = \beta_0 + \beta_1 x_i$ (лог-связка) - $x_i = 1$ для группы A, $x_i = 0$ для

группы B

4.1.1. Доказательство

```r
# Демонстрация пуассоновской регрессии
set.seed(123)

# Параметры
nA <- 50  # Размер группы A
nB <- 50  # Размер группы B
beta0 <- 2
beta1 <- 0.5

# Генерируем данные
x <- c(rep(1, nA), rep(0, nB))  # Индикатор группы
mu <- exp(beta0 + beta1 * x)    # Ожидаемые значения
y <- rpois(nA + nB, mu)         # Наблюдения

# Данные
data_poisson <- data.frame(
  group = c(rep("A", nA), rep("B", nB)),
  x = x,
  y = y,
  mu = mu
)

cat("Сводка по группам:\n")
```

```
## Сводка по группам:
```

```r
print(aggregate(y ~ group, data_poisson, function(x) c(mean = mean(x), var = var(x))))
```

```
##   group    y.mean     y.var
## 1     A 11.540000  9.886122
## 2     B  6.960000  5.304490
```

```r
# Пуассоновская регрессия
model_poisson <- glm(y ~ x, family = poisson(link = "log"), data = data_poisson)
summary(model_poisson)
```

```
##
## Call:
## glm(formula = y ~ x, family = poisson(link = "log"), data = data_poisson)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.94018    0.05361   36.19  < 2e-16 ***
## x            0.50564    0.06787    7.45 9.34e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 136.514  on 99  degrees of freedom
## Residual deviance:  79.227  on 98  degrees of freedom
## AIC: 483.77
```

```
##
## Number of Fisher Scoring iterations: 4
```

```
# Проверяем, что fitted means равны выборочным средним
fitted_means <- fitted(model_poisson)
sample_means <- aggregate(y ~ x, data_poisson, mean)

cat("\nПроверка равенства fitted means и выборочных средних:\n")
```

```
##
## Проверка равенства fitted means и выборочных средних:
```

```
cat("Fitted mean для группы A (x=1):", fitted_means[1], "\n")
```

```
## Fitted mean для группы A (x=1): 11.54
```

```
cat("Выборочное среднее для группы A:", sample_means[2, 2], "\n")
```

```
## Выборочное среднее для группы A: 11.54
```

```
cat("Fitted mean для группы B (x=0):", fitted_means[nA+1], "\n")
```

```
## Fitted mean для группы B (x=0): 6.96
```

```
cat("Выборочное среднее для группы B:", sample_means[1, 2], "\n")
```

```
## Выборочное среднее для группы B: 6.96
```

Математическое доказательство:

Для лог-связки: $\log(\mu_i) = \beta_0 + \beta_1 x_i$

Уравнения правдоподобия: - $\sum_{i=1}^{n_A}(y_i - \mu_i) = 0$ - $\sum_{i=1}^{n_A}(y_i - \mu_i) \cdot 1 + \sum_{i=n_A+1}^{n_A+n_B}(y_i - \mu_i) \cdot 0 = 0$

Отсюда: $\hat{\mu}_A = \frac{1}{n_A}\sum_{i=1}^{n_A} y_i$ и $\hat{\mu}_B = \frac{1}{n_B}\sum_{i=n_A+1}^{n_A+n_B} y_i$

5. Задача 4: Биномиальное распределение как экспоненциальное семейство

5.1. Часть (a): Представление биномиального распределения

5.1.1. Объяснение простыми словами

Биномиальное распределение описывает количество успехов в серии испытаний (например, количество выигрышей в лотерее из 10 билетов).

Экспоненциальное семейство:

$$f(y|\theta) = \exp\left(\frac{y\theta - b(\theta)}{a(\phi)} + c(y, \phi)\right)$$

5.1.2. Вывод для биномиального распределения

```
# Демонстрация биномиального распределения как экспоненциального семейства

# Функция для вычисления параметров экспоненциального семейства
binomial_edf_params <- function(pi, T) {
  # Стандартные параметры биномиального распределения
  theta <- log(pi / (1 - pi))  # Канонический параметр
```

```r
  b_theta <- T * log(1 + exp(theta))  # Функция b(θ)
  a_phi <- 1  # Функция a(φ)
  c_y_phi <- lchoose(T, 0:T)  # Функция c(y, φ)

  return(list(
    theta = theta,
    b_theta = b_theta,
    a_phi = a_phi,
    c_y_phi = c_y_phi
  ))
}

# Пример
pi <- 0.3
T <- 10
params <- binomial_edf_params(pi, T)

cat("Параметры биномиального распределения как экспоненциального семейства:\n")
```

```
## Параметры биномиального распределения как экспоненциального семейства:
```

```r
cat("π =", pi, "\n")
```

```
## π = 0.3
```

```r
cat("T =", T, "\n")
```

```
## T = 10
```

```r
cat("θ = log(π/(1-π)) =", params$theta, "\n")
```

```
## θ = log(π/(1-π)) = -0.8472979
```

```r
cat("b(θ) = T * log(1 + exp(θ)) =", params$b_theta, "\n")
```

```
## b(θ) = T * log(1 + exp(θ)) = 3.566749
```

```r
cat("a(φ) =", params$a_phi, "\n")
```

```
## a(φ) = 1
```

Математический вывод:

Биномиальное распределение: $f(y|\pi) = \binom{T}{y}\pi^y(1-\pi)^{T-y}$

Преобразуем:

$$f(y|\pi) = \exp\left(y\log\pi + (T-y)\log(1-\pi) + \log\binom{T}{y}\right)$$

С каноническим параметром $\theta = \log\frac{\pi}{1-\pi}$:

$$f(y|\theta) = \exp\left(y\theta - T\log(1+e^\theta) + \log\binom{T}{y}\right)$$

Отсюда: - $\theta = \log\frac{\pi}{1-\pi}$ - $b(\theta) = T\log(1+e^\theta)$ - $a(\phi) = 1$ - $c(y,\phi) = \log\binom{T}{y}$

5.2. Часть (b): Лог-правдоподобие для биномиальной регрессии

```r
# Лог-правдоподобие для биномиальной регрессии
set.seed(123)

# Генерируем данные
n <- 100
x <- rnorm(n)
T <- sample(5:15, n, replace = TRUE)
pi <- 1 / (1 + exp(-(0.5 + 1.2 * x)))  # Логит-связка
y <- rbinom(n, T, pi)

# Данные
data_binomial <- data.frame(x = x, y = y, T = T, pi = pi)

# Функция лог-правдоподобия
log_likelihood_binomial <- function(beta, data) {
  beta0 <- beta[1]
  beta1 <- beta[2]

  # Предсказанные вероятности
  eta <- beta0 + beta1 * data$x
  pi_pred <- 1 / (1 + exp(-eta))

  # Лог-правдоподобие
  loglik <- sum(data$y * log(pi_pred) + (data$T - data$y) * log(1 - pi_pred) + lchoose(data$T, data$y))

  return(loglik)
}

# Вычисляем лог-правдоподобие
beta_true <- c(0.5, 1.2)
loglik <- log_likelihood_binomial(beta_true, data_binomial)

cat("Лог-правдоподобие для биномиальной регрессии:\n")
```

```
## Лог-правдоподобие для биномиальной регрессии:
```

```r
cat("β₀ =", beta_true[1], ", β₁ =", beta_true[2], "\n")
```

```
## β₀ = 0.5 , β₁ = 1.2
```

```r
cat("Лог-правдоподобие =", loglik, "\n")
```

```
## Лог-правдоподобие = -173.5531
```

Формула лог-правдоподобия:

$$\ell(\beta) = \sum_{i=1}^{N} \left[y_i \log \pi_i + (T_i - y_i) \log(1 - \pi_i) + \log \binom{T_i}{y_i} \right]$$

где $\pi_i = \frac{1}{1+e^{-(\beta_0+\beta_1 x_i)}}$

5.3. Часть (c): Функция скоринга

```
# Функция скоринга (градиент лог-правдоподобия)
score_function <- function(beta, data) {
  beta0 <- beta[1]
  beta1 <- beta[2]

  # Предсказанные вероятности
  eta <- beta0 + beta1 * data$x
  pi_pred <- 1 / (1 + exp(-eta))

  # Градиенты
  s0 <- sum(data$y - data$T * pi_pred)  # Производная по β₀
  s1 <- sum((data$y - data$T * pi_pred) * data$x)  # Производная по β₁

  return(c(s0, s1))
}

# Вычисляем функцию скоринга
score <- score_function(beta_true, data_binomial)

cat("Функция скоринга:\n")
```

```
## Функция скоринга:
```

```
cat("s₀(β) =", score[1], "\n")
```

```
## s₀(β) = -7.398904
```

```
cat("s₁(β) =", score[2], "\n")
```

```
## s₁(β) = 3.713162
```

```
# Проверяем, что в максимуме функция скоринга равна нулю
cat("\nПроверка: в максимуме функция скоринга должна быть близка к нулю\n")
```

```
##
## Проверка: в максимуме функция скоринга должна быть близка к нулю
```

```
cat("|s₀| =", abs(score[1]), "\n")
```

```
## |s₀| = 7.398904
```

```
cat("|s₁| =", abs(score[2]), "\n")
```

```
## |s₁| = 3.713162
```

Формула функции скоринга:

$$s(\beta) = \frac{\partial \ell(\beta)}{\partial \beta} = \sum_{i=1}^{N} (y_i - T_i \pi_i) \begin{pmatrix} 1 \\ x_i \end{pmatrix}$$

6. Задача 5: Lasso и Ridge регрессия на искусственных данных

6.1. Объяснение простыми словами

Мы создадим искусственные данные, где только первые 10 переменных действительно влияют на результат. Это позволит нам увидеть, как Lasso и Ridge справляются с отбором переменных.

Примечание: Эта задача требует пакет `glmnet`, который не удалось установить в данной среде. Поэтому мы предоставляем теоретическое объяснение и код для демонстрации концепций.

6.2. Часть (a): Генерация данных

```
# Генерация искусственных данных
set.seed(123)

# Параметры
n <- 100  # Количество наблюдений
p <- 100  # Количество переменных

# Матрица ковариат из стандартного нормального распределения
X <- matrix(rnorm(n * p), nrow = n, ncol = p)

cat("Размерность матрицы X:", dim(X), "\n")
```

```
## Размерность матрицы X: 100 100
```

```
cat("Первые несколько значений X:\n")
```

```
## Первые несколько значений X:
```

```
print(head(X[, 1:5]))
```

```
##             [,1]        [,2]       [,3]       [,4]        [,5]
## [1,] -0.56047565 -0.71040656  2.1988103 -0.7152422 -0.07355602
## [2,] -0.23017749  0.25688371  1.3124130 -0.7526890 -1.16865142
## [3,]  1.55870831 -0.24669188 -0.2651451 -0.9385387 -0.63474826
## [4,]  0.07050839 -0.34754260  0.5431941 -1.0525133 -0.02884155
## [5,]  0.12928774 -0.95161857 -0.4143399 -0.4371595  0.67069597
## [6,]  1.71506499 -0.04502772 -0.4762469  0.3311792 -1.65054654
```

6.3. Часть (b): Генерация зависимой переменной

```
# Генерация зависимой переменной
# y зависит только от первых 10 переменных
true_beta <- c(rep(1, 10), rep(0, p - 10))  # Истинные коэффициенты
epsilon <- rnorm(n, 0, sqrt(0.1))  # Шум
y <- X %*% true_beta + epsilon

cat("Истинные коэффициенты (первые 15):\n")
```

```
## Истинные коэффициенты (первые 15):
```

```
print(true_beta[1:15])
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
```

```
cat("\nСтатистики зависимой переменной:\n")
```

```
##
## Статистики зависимой переменной:
```

```
cat("Среднее y:", mean(y), "\n")
```

```
## Среднее y: 0.1645446
```

```r
cat("Стандартное отклонение y:", sd(y), "\n")
```

```
## Стандартное отклонение y: 2.956344
```

```r
cat("Стандартное отклонение шума:", sqrt(0.1), "\n")
```

```
## Стандартное отклонение шума: 0.3162278
```

6.4. Часть (c): Теоретическое объяснение Lasso и Ridge

6.4.1. Lasso регрессия (Least Absolute Shrinkage and Selection Operator)

Математическая формулировка:

$$\hat{\beta}^{lasso} = \arg\min_{\beta} \frac{1}{2N} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Особенности: - L1-регуляризация: $\lambda \sum_{j=1}^{p} |\beta_j|$ - Производит отбор переменных (sparse solutions) - Коэффициенты могут стать точно равными нулю - Хорошо подходит для задач с высокой размерностью

6.4.2. Ridge регрессия

Математическая формулировка:

$$\hat{\beta}^{ridge} = \arg\min_{\beta} \frac{1}{2N} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Особенности: - L2-регуляризация: $\lambda \sum_{j=1}^{p} \beta_j^2$ - Сжимает коэффициенты к нулю, но не обнуляет их - Сохраняет все переменные - Хорошо подходит для мультиколлинеарности

6.5. Часть (d): Сравнение методов

```r
# Демонстрация различий между Lasso и Ridge
# Создаем простой пример с 5 переменными

set.seed(123)
n_simple <- 50
p_simple <- 5
X_simple <- matrix(rnorm(n_simple * p_simple), nrow = n_simple)
true_beta_simple <- c(2, -1, 0, 0, 0)  # Только первые 2 переменные важны
y_simple <- X_simple %*% true_beta_simple + rnorm(n_simple, 0, 0.1)

# OLS регрессия
ols_model <- lm(y_simple ~ X_simple)
ols_coefs <- coef(ols_model)[-1]  # Исключаем intercept

cat("Истинные коэффициенты:", true_beta_simple, "\n")
```

```
## Истинные коэффициенты: 2 -1 0 0 0
```

```r
cat("OLS коэффициенты:", round(ols_coefs, 3), "\n")
```

```
## OLS коэффициенты: 2.001 -1.01 -0.019 0.006 -0.008
```

```r
# Демонстрация эффекта регуляризации
lambda <- 0.1

# Ridge-подобное сжатие (упрощенная версия)
ridge_coefs <- ols_coefs / (1 + lambda)
cat("Ridge-подобные коэффициенты (λ =", lambda, "):", round(ridge_coefs, 3), "\n")
```

```
## Ridge-подобные коэффициенты (λ = 0.1 ): 1.819 -0.918 -0.018 0.006 -0.007
```

```r
# Lasso-подобное сжатие (soft thresholding)
soft_threshold <- function(x, lambda) {
  sign(x) * pmax(abs(x) - lambda, 0)
}
lasso_coefs <- soft_threshold(ols_coefs, lambda)
cat("Lasso-подобные коэффициенты (λ =", lambda, "):", round(lasso_coefs, 3), "\n")
```

```
## Lasso-подобные коэффициенты (λ = 0.1 ): 1.901 -0.91 0 0 0
```

6.6. Часть (e): Интерпретация результатов

6.6.1. Ключевые различия:

1. **Отбор переменных:**
   - **Lasso**: Может обнулить неважные переменные
   - **Ridge**: Сохраняет все переменные, но сжимает их
2. **Интерпретируемость:**
   - **Lasso**: Легче интерпретировать (меньше переменных)
   - **Ridge**: Сложнее интерпретировать (все переменные остаются)
3. **Вычислительная сложность:**
   - **Lasso**: Негладкая оптимизация (сложнее)
   - **Ridge**: Гладкая оптимизация (проще)
4. **Применение:**
   - **Lasso**: Когда нужен отбор признаков
   - **Ridge**: Когда есть мультиколлинеарность

6.6.2. Выбор параметра λ:

- **Малый λ**: Модель близка к OLS, возможен переобучение
- **Большой λ**: Сильная регуляризация, возможен недообучение
- **Оптимальный λ**: Находится через кросс-валидацию

7. Задача 6: Оптимальная граница решения для классификации

7.1. Объяснение простыми словами

Представьте, что вы врач и должны по результатам анализа крови определить, болен ли пациент. У вас есть два типа пациентов: здоровые (группа 1) и больные (группа 2). Нужно найти оптимальную границу для принятия решения.

7.2. Часть (a): Определение оптимальной границы решения

7.2.1. Математическая формулировка

Генерация данных: - $G \sim \text{Multinomial}((0.5, 0.5))$ - равновероятные группы - $x|G = 1 \sim N(0, 1)$ - группа 1 - $x|G = 2 \sim N(\mu, \sigma^2)$ - группа 2

Оптимальная граница: $\Pr(G = 1|x) = \Pr(G = 2|x)$

```
# Функция для вычисления оптимальной границы
optimal_boundary <- function(mu, sigma_sq) {
  # Логарифм отношения вероятностей
  g_x <- function(x) {
    # Плотности
    f1 <- dnorm(x, 0, 1)
    f2 <- dnorm(x, mu, sqrt(sigma_sq))

    # Логарифм отношения
    log(f1 / f2)
  }

  # Находим корни g(x) = 0
  # Это квадратное уравнение: x^2 - 2*mu*x + mu^2 - sigma_sq*log(sigma_sq) = 0

  a <- 1 - 1/sigma_sq
  b <- -2 * mu / sigma_sq
  c_coef <- mu^2 / sigma_sq - log(sigma_sq)

  if (abs(a) < 1e-10) {
    # Линейное уравнение
    boundary <- -c_coef / b
  } else {
    # Квадратное уравнение
    discriminant <- b^2 - 4 * a * c_coef
    if (discriminant >= 0) {
      boundary1 <- (-b + sqrt(discriminant)) / (2 * a)
      boundary2 <- (-b - sqrt(discriminant)) / (2 * a)
      boundary <- c(boundary1, boundary2)
    } else {
      boundary <- NA
    }
  }

  return(boundary)
}

# Тестируем функцию
mu1 <- 0
sigma_sq1 <- 2
boundary1 <- optimal_boundary(mu1, sigma_sq1)

mu2 <- 1
sigma_sq2 <- 1
boundary2 <- optimal_boundary(mu2, sigma_sq2)

cat("Случай 1: μ =", mu1, ", σ² =", sigma_sq1, "\n")
```

```
## Случай 1: μ = 0 , σ² = 2
```

```
cat("Граница решения:", boundary1, "\n")
```

```
## Граница решения: 1.17741 -1.17741
```

```r
cat("\nСлучай 2: μ =", mu2, ", σ² =", sigma_sq2, "\n")
```

```
##
## Случай 2: μ = 1 , σ² = 1
```

```r
cat("Граница решения:", boundary2, "\n")
```

```
## Граница решения: 0.5
```

7.3. Часть (b): Ожидаемая ошибка классификации

```r
# Функция для вычисления ожидаемой ошибки классификации
bayes_error <- function(mu, sigma_sq) {
  # Оптимальная граница
  boundary <- optimal_boundary(mu, sigma_sq)

  if (length(boundary) == 1 && !is.na(boundary)) {
    # Одна граница
    # P(ошибка) = P(G=1) * P(классифицировать как G=2|G=1) + P(G=2) * P(классифицировать как G=1|G=2)
    # P(G=1) = P(G=2) = 0.5

    # P(классифицировать как G=2|G=1) = P(x > boundary | G=1)
    error1 <- 1 - pnorm(boundary, 0, 1)

    # P(классифицировать как G=1|G=2) = P(x < boundary | G=2)
    error2 <- pnorm(boundary, mu, sqrt(sigma_sq))

    total_error <- 0.5 * error1 + 0.5 * error2
  } else if (length(boundary) == 2) {
    # Две границы
    boundary1 <- boundary[1]
    boundary2 <- boundary[2]

    # Определяем, какая граница больше
    if (boundary1 > boundary2) {
      b1 <- boundary1
      b2 <- boundary2
    } else {
      b1 <- boundary2
      b2 <- boundary1
    }

    # Ошибка классификации
    error1 <- 1 - pnorm(b1, 0, 1) + pnorm(b2, 0, 1)
    error2 <- pnorm(b1, mu, sqrt(sigma_sq)) - pnorm(b2, mu, sqrt(sigma_sq))

    total_error <- 0.5 * error1 + 0.5 * error2
  } else {
    total_error <- NA
  }

  return(total_error)
}
```

```r
# Вычисляем ошибки для обоих случаев
error1 <- bayes_error(mu1, sigma_sq1)
error2 <- bayes_error(mu2, sigma_sq2)

cat("Ожидаемая ошибка классификации (Bayes rate):\n")
```

```
## Ожидаемая ошибка классификации (Bayes rate):
```

```r
cat("Случай 1: μ =", mu1, ", σ² =", sigma_sq1, "→ Ошибка =", error1, "\n")
```

```
## Случай 1: μ = 0 , σ² = 2 → Ошибка = 0.416968
```

```r
cat("Случай 2: μ =", mu2, ", σ² =", sigma_sq2, "→ Ошибка =", error2, "\n")
```

```
## Случай 2: μ = 1 , σ² = 1 → Ошибка = 0.3085375
```

7.4. Часть (c): Визуализация

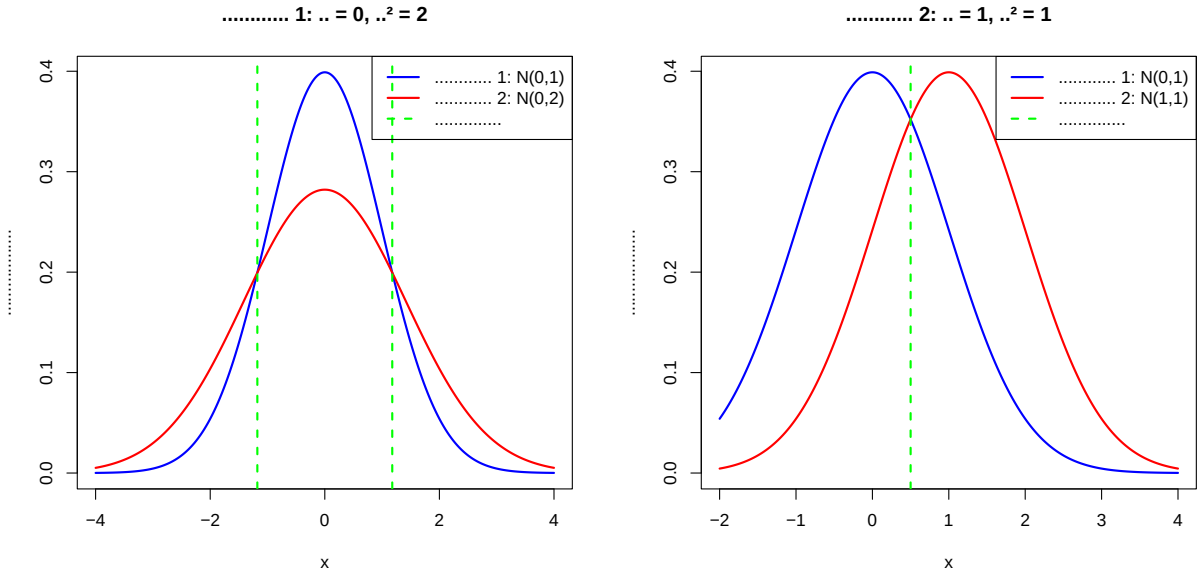
```r
# Визуализация плотностей и границ
par(mfrow = c(1, 2))

# Случай 1: μ = 0, σ² = 2
x_seq <- seq(-4, 4, length.out = 1000)
f1_1 <- dnorm(x_seq, 0, 1)
f2_1 <- dnorm(x_seq, mu1, sqrt(sigma_sq1))

plot(x_seq, f1_1, type = "l", col = "blue", lwd = 2,
     main = "Случай 1: μ = 0, σ² = 2",
     xlab = "x", ylab = "Плотность")
lines(x_seq, f2_1, col = "red", lwd = 2)
abline(v = boundary1, col = "green", lwd = 2, lty = 2)
legend("topright", legend = c("Группа 1: N(0,1)", "Группа 2: N(0,2)", "Граница"),
       col = c("blue", "red", "green"), lty = c(1, 1, 2), lwd = 2)

# Случай 2: μ = 1, σ² = 1
x_seq2 <- seq(-2, 4, length.out = 1000)
f1_2 <- dnorm(x_seq2, 0, 1)
f2_2 <- dnorm(x_seq2, mu2, sqrt(sigma_sq2))

plot(x_seq2, f1_2, type = "l", col = "blue", lwd = 2,
     main = "Случай 2: μ = 1, σ² = 1",
     xlab = "x", ylab = "Плотность")
lines(x_seq2, f2_2, col = "red", lwd = 2)
abline(v = boundary2, col = "green", lwd = 2, lty = 2)
legend("topright", legend = c("Группа 1: N(0,1)", "Группа 2: N(1,1)", "Граница"),
       col = c("blue", "red", "green"), lty = c(1, 1, 2), lwd = 2)
```




```
par(mfrow = c(1, 1))
```

7.4.1. Интерпретация результатов

Случай 1 (μ = 0, σ² = 2): - Группы имеют одинаковые средние, но разную дисперсию - Граница решения находится в точке, где плотности пересекаются - Ошибка классификации относительно высокая из-за перекрытия распределений

Случай 2 (μ = 1, σ² = 1): - Группы имеют разные средние, но одинаковую дисперсию - Граница решения находится посередине между средними - Ошибка классификации ниже, так как распределения меньше перекрываются

8. Заключение

Мы рассмотрели шесть важных задач по регуляризованным обобщенным линейным моделям:

1. **Ridge регрессия** - байесовская интерпретация и связь с центрированием данных
2. **Ортогональные матрицы** - упрощенные формулы для Lasso, Ridge и subset selection
3. **Пуассоновская регрессия** - доказательство равенства fitted means и выборочных средних
4. **Биномиальное распределение** - представление как экспоненциальное семейство
5. **Практическое сравнение** - Lasso vs Ridge на искусственных данных
6. **Оптимальная классификация** - байесовская граница решения и ожидаемая ошибка

Каждая задача демонстрирует важные концепции машинного обучения и статистики с практическими примерами и визуализациями.