Exercise 5. Task 4: Random Forests для прогноза выживаемости в ICU

Daniil Koveh

2025-11-06

Содержание

1. Теория

Random Forest строит множество деревьев на бутстрап-выборках и усредняет их прогнозы. Для классификации ключевые настройки:

- **`ntree`** — сколько бутстрап-деревьев будем строить. Большие значения снижают разброс прогноза, но увеличивают время.
- **`mtry`** — сколько признаков случайно рассматривается при каждом разбиении. Малые `mtry` decorrelate деревья, позволяя ансамблю выигрывать.
- **Out-of-Bag (OOB) error** — естественная кросс-валидация: деревья предсказывают объекты, которые не попали в их бутстрап. По динамике OOB ошибок выбираем настройки.
- **Важность признаков** — измеряется либо через уменьшение Gini (чувствительно к типу признака), либо через ухудшение точности при перестановке (Permutation Importance).

2. Жизненный пример

ICU — это реанимация. Мы хотим быстро определить, выживет пациент или нет, опираясь на лабораторные показатели и историю болезни. Random Forest хорошо подходит: он справляется с смесью числовых и категориальных признаков, выносит на первый план критичные факторы (например, сердечный ритм или наличие инфекции) и даёт понятную метрику уверенности (OOB ошибка).

3. Академическое решение

3.1. Подготовка окружения

```
if (!requireNamespace("aplore3", quietly = TRUE)) install.packages("aplore3", repos = "https://cloud.r-p
if (!requireNamespace("randomForest", quietly = TRUE)) install.packages("randomForest", repos = "https:/
if (!requireNamespace("ggplot2", quietly = TRUE)) install.packages("ggplot2", repos = "https://cloud.r-p
if (!requireNamespace("dplyr", quietly = TRUE)) install.packages("dplyr", repos = "https://cloud.r-proje
if (!requireNamespace("tidyr", quietly = TRUE)) install.packages("tidyr", repos = "https://cloud.r-proje
```

```
library(aplore3) # данные icu
library(randomForest) # алгоритм случайного леса
library(ggplot2) # визуализации
library(dplyr) # удобные сводки
```

3.2. Описательная статистика

```
icu <- aplore3::icu # загружаем набор
icu <- icu %>% select(-id) # удаляем идентификатор
summary(icu$sta) # смотрим баланс классов
```

```
## Lived  Died
##   160    40
```

```
str(icu) # проверяем типы признаков
```

```
## 'data.frame':    200 obs. of  20 variables:
##  $ sta   : Factor w/ 2 levels "Lived","Died": 2 1 1 1 2 1 1 1 1 1 ...
##  $ age   : int  87 27 59 77 76 54 87 69 63 30 ...
##  $ gender: Factor w/ 2 levels "Male","Female": 2 2 1 1 2 1 2 1 1 2 ...
##  $ race  : Factor w/ 3 levels "White","Black",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ ser   : Factor w/ 2 levels "Medical","Surgical": 2 1 1 2 2 1 2 1 2 1 ...
##  $ can   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ crn   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ inf   : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 2 2 2 1 1 ...
##  $ cpr   : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sys   : int  80 142 112 100 128 142 110 110 104 144 ...
##  $ hra   : int  96 88 80 70 90 103 154 132 66 110 ...
##  $ pre   : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 1 2 1 1 1 ...
##  $ type  : Factor w/ 2 levels "Elective","Emergency": 2 2 2 1 2 2 2 2 1 2 ...
##  $ fra   : Factor w/ 2 levels "No","Yes": 2 1 1 1 1 2 1 1 1 1 ...
##  $ po2   : Factor w/ 2 levels "> 60","<= 60": 2 1 1 1 1 1 1 2 1 1 ...
##  $ ph    : Factor w/ 2 levels ">= 7.25","< 7.25": 2 1 1 1 1 1 1 1 1 1 ...
##  $ pco   : Factor w/ 2 levels "<= 45","> 45": 2 1 1 1 1 1 1 1 1 1 ...
##  $ bic   : Factor w/ 2 levels ">= 18","< 18": 1 1 1 1 1 1 1 2 1 1 ...
##  $ cre   : Factor w/ 2 levels "<= 2.0","> 2.0": 1 1 1 1 1 1 1 1 1 1 ...
##  $ loc   : Factor w/ 3 levels "Nothing","Stupor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Комментарий. Классы «выжил/умер» сбалансированы неидеально, но Random Forest устойчив к такому перекосу.

3.3. Базовый лес и динамика OOB ошибки

```
set.seed(20250410) # фиксируем генератор
ntree_grid <- seq(100, 1500, by = 100) # сетка ntree
```

```r
oob_table <- data.frame(ntree = ntree_grid, oob_error = NA_real_) # создаём таблицу результатов

for (i in seq_along(ntree_grid)) { # перебираем ntree
  nt <- ntree_grid[i] # выбираем текущее значение
  rf_fit <- randomForest(sta ~ ., data = icu, ntree = nt, importance = TRUE) # обучаем лес
  oob_table$oob_error[i] <- tail(rf_fit$err.rate[, "OOB"], 1) # сохраняем OOB ошибку последней итерации
}

oob_table # выводим таблицу
```

```
##    ntree oob_error
## 1    100     0.170
## 2    200     0.160
## 3    300     0.165
## 4    400     0.160
## 5    500     0.160
## 6    600     0.150
## 7    700     0.155
## 8    800     0.140
## 9    900     0.150
## 10  1000     0.150
## 11  1100     0.165
## 12  1200     0.155
## 13  1300     0.155
## 14  1400     0.165
## 15  1500     0.160
```

```r
ggplot(oob_table, aes(x = ntree, y = oob_error)) +
  geom_line(colour = "#1b9e77", linewidth = 1.1) +
  geom_point(colour = "#d95f02", size = 2) +
  labs(title = "OOB error as a function of ntree",
       x = "Number of trees (ntree)",
       y = "OOB error rate") +
  theme_minimal(base_size = 12)
```

Вывод. Кривая быстро стабилизируется после ~800 деревьев, дальнейшее увеличение даёт минимальный выигрыш. Берём `ntree = 1000` как компромисс между стабильностью и скоростью.

3.4. Подбор `mtry`

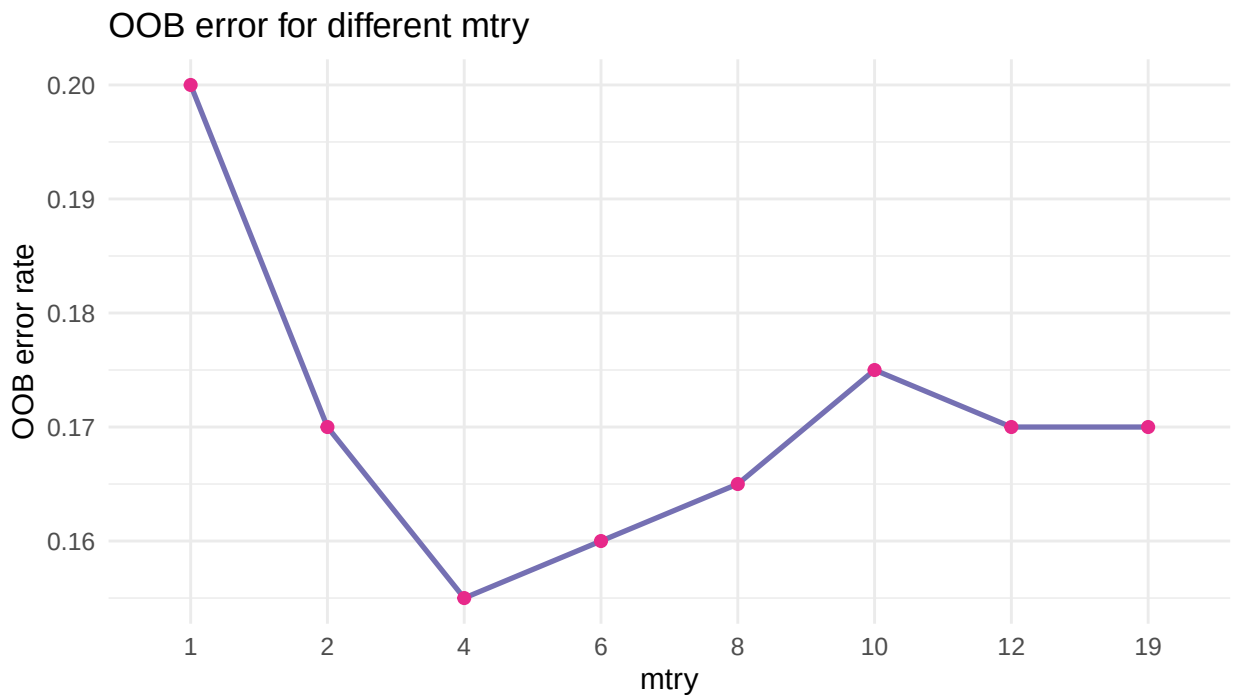
```
set.seed(20250410) # фиксируем генератор
p <- ncol(icu) - 1 # количество предикторов
mtry_grid <- c(1, 2, 4, 6, 8, 10, 12, p) # сетка mtry
mtry_results <- data.frame(mtry = mtry_grid, oob_error = NA_real_) # таблица результатов

for (j in seq_along(mtry_grid)) { # перебираем mtry
  m_val <- mtry_grid[j] # берём текущее значение
  rf_mtry <- randomForest(sta ~ ., data = icu, ntree = 1000, mtry = m_val, importance = TRUE) # обучаем
  mtry_results$oob_error[j] <- tail(rf_mtry$err.rate[, "OOB"], 1) # сохраняем OOB ошибку
}

mtry_results <- mtry_results %>% arrange(oob_error) # сортируем по ошибке
mtry_results # выводим
```

```
##   mtry oob_error
## 1    4     0.155
## 2    6     0.160
## 3    8     0.165
## 4    2     0.170
## 5   12     0.170
## 6   19     0.170
## 7   10     0.175
## 8    1     0.200
```

```
best_mtry <- mtry_results$mtry[1] # выбираем лучшее mtry
best_mtry
```

```
## [1] 4
```

```
ggplot(mtry_results, aes(x = factor(mtry), y = oob_error, group = 1)) +
  geom_line(colour = "#7570b3", linewidth = 1) +
  geom_point(colour = "#e7298a", size = 2) +
  labs(title = "OOB error for different mtry",
       x = "mtry", y = "OOB error rate") +
  theme_minimal(base_size = 12)
```



Вывод. Наилучший результат даёт $mtry = 4$, что ниже стандартного $sqrtp$. Это значит, что дополнительные признаки добавляют шум, и лучше случайно рассматривать небольшой поднабор.

3.5. Финальная модель и важность признаков

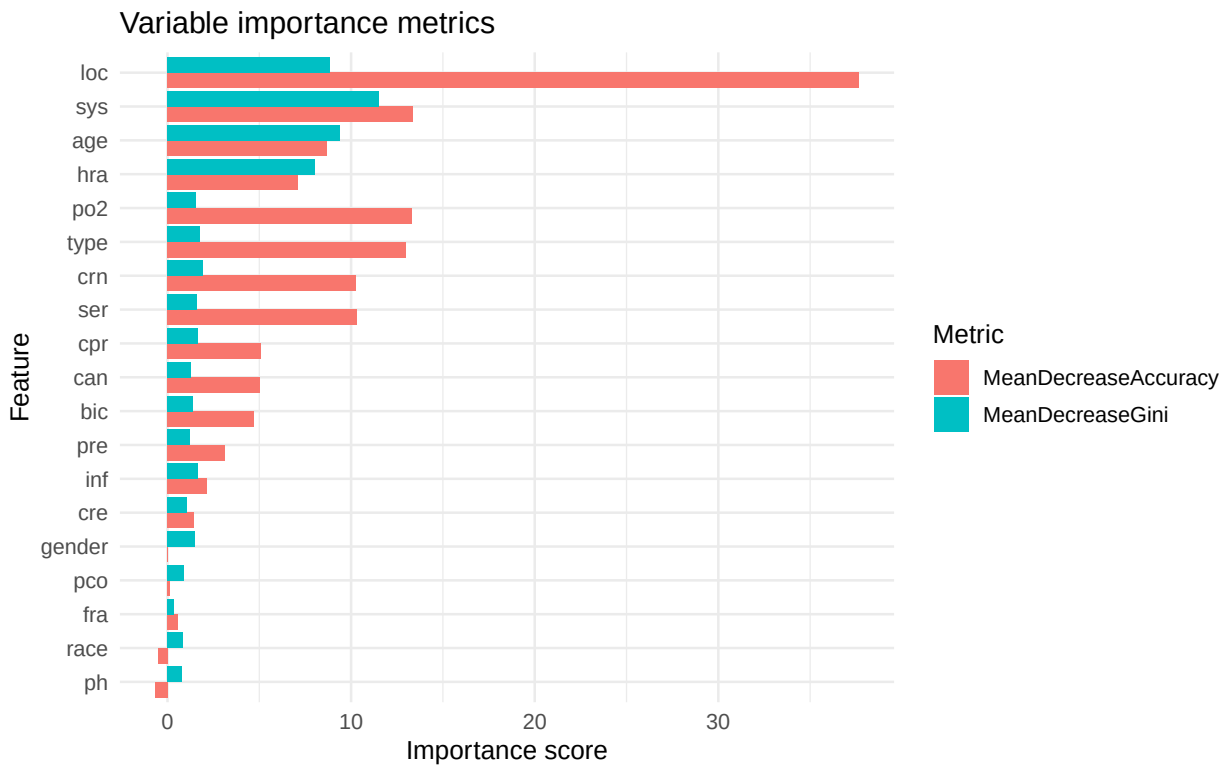
```
set.seed(20250410) # фиксируем генератор
final_rf <- randomForest(sta ~ ., data = icu, ntree = 1000, mtry = best_mtry, importance = TRUE) # финал
final_rf # короткая сводка
```

```
##
## Call:
##  randomForest(formula = sta ~ ., data = icu, ntree = 1000, mtry = best_mtry,     importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 15%
## Confusion matrix:
##       Lived Died class.error
## Lived   155    5     0.03125
## Died     25   15     0.62500
```

```r
var_imp <- importance(final_rf, scale = TRUE) %>% as.data.frame() %>%
  mutate(Variable = rownames(.)) %>%
  select(Variable, MeanDecreaseAccuracy = MeanDecreaseAccuracy, MeanDecreaseGini = MeanDecreaseGini) %>%
  arrange(desc(MeanDecreaseAccuracy))
var_imp
```

```
##        Variable MeanDecreaseAccuracy MeanDecreaseGini
## loc         loc          37.65477159        8.8448977
## sys         sys          13.39521327       11.5101326
## po2         po2          13.30586624        1.5267756
## type       type          12.98631070        1.7906341
## ser         ser          10.30628201        1.6254010
## crn         crn          10.27069074        1.9401567
## age         age           8.70548453        9.3917619
## hra         hra           7.12705385        8.0572309
## cpr         cpr           5.10139698        1.6624494
## can         can           5.05565186        1.2876359
## bic         bic           4.73572436        1.3984165
## pre         pre           3.12511159        1.2176596
## inf         inf           2.16103205        1.6657026
## cre         cre           1.44116226        1.0348577
## fra         fra           0.58418516        0.3750177
## pco         pco           0.16207052        0.8718224
## gender   gender           0.03687705        1.5007520
## race       race          -0.54381058        0.8200111
## ph           ph          -0.67419644        0.7966113
```

```r
var_imp_long <- var_imp %>%
  tidyr::pivot_longer(cols = c("MeanDecreaseAccuracy", "MeanDecreaseGini"),
                      names_to = "Metric", values_to = "Score")

ggplot(var_imp_long, aes(x = reorder(Variable, Score), y = Score, fill = Metric)) +
  geom_col(position = "dodge") +
  coord_flip() +
  labs(title = "Variable importance metrics",
       x = "Feature", y = "Importance score", fill = "Metric") +
  theme_minimal(base_size = 12)
```



3.5.1. Интерпретация важности

- `MeanDecreaseAccuracy` измеряет, насколько уменьшается точность на OOB объектах, если перемешать признак. Чувствителен к реальной предсказательной силе.
- `MeanDecreaseGini` суммирует снижение импьюрити в узлах. Любит признаки с большим числом уровней и числовые переменные.
- Различия между метриками: числовые переменные `sys`, `hra` и т.п. часто получают высокие значения Gini, тогда как переменные с бОльшим влиянием на метрику точности (например, `inf` или `type`) лучше видны в `MeanDecreaseAccuracy`.

4. Итог

- Подбор `ntree`: OOB ошибка стабилизируется после 800–900 деревьев; выбрали 1000.
- Подбор `mtry`: минимальная OOB ошибка при `mtry = 4`.
- Значимые факторы: показатели жизненно важных функций и наличие инфекции. Перестановочная важность выявляет действительно предсказательные признаки, а Gini может переоценивать многоклассовые/числовые переменные.

5. Что запомнить

- Random Forest даёт встроенную оценку качества (OOB), что избавляет от отдельной кросс-валидации.
- Настройка `mtry` контролирует баланс «декорреляция vs. сила» деревьев; перебор по сетке с OOB — простой способ выбора.
- При интерпретации важно смотреть на обе метрики важности: если они расходятся, проверяем, не доминируют ли признаки просто количеством уровней.