

## Лабораторна 7: Дерева рішень.

### Дерева класифікації

Бібліотека tree використовується для побудови дерев класифікації та регресії.

```
> library(tree)
```

Спочатку ми розглянемо дерева класифікації для аналізу набору даних Carseats. У цих даних, Sales - це неперервна змінна, і тому ми починаємо з перекодування її на двійкову. Ми використаємо функцію ifelse () для створення змінної, яка називається High та приймає значення Yes, якщо значення змінної Sales перевищує 8 та приймає значення No інакше.

```
> library(ISLR)
> attach(Carseats)
> High=ifelse(Sales <=8 , "No" , "Yes")
```

Нарешті, ми використаємо функцію data.frame () для об'єднання High з рештою даних Carseats.

```
> Carseats=data.frame(Carseats,High)
```

Тепер ми використаємо функцію tree () для підбору дерева класифікації для прогнозування High, використовуючи всі змінні, крім Sales. Синтаксис функції tree () подібний до синтаксису функції lm().

```
> tree.carseats=tree(High~. - Sales , Carseats)
```

Функція summary () перелічує змінні, які використовуються як внутрішні вузли у дереві, кількість термінальних вузлів та навчальну частоту помилок.

```
> summary(tree.carseats)
```

```
Classification tree:
tree(formula = High ~ . - Sales , data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"      "Price"          "Income"         "CompPrice"
[5] "Population"     "Advertising"    "Age"           "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

Ми бачимо, що рівень помилок на навчальних даних становить 9%. Для класифікаційних дерев - відхилення наведене `summary()` задається наступним чином

$$-2 \sum_m \sum_k n_{mk} \log \hat{p}_{mk}$$

де  $n_{mk}$  - кількість спостережень у  $m$ -му кінцевому вузлі, що належать до  $k$ -го класу. Невелике відхилення вказує, що дерево добре підходить до навчальних даних. Зафіксована середня девіація залишків дорівнює девіації, поділеній на  $n - |T_0|$ , що в цьому випадку дорівнює  $400-27 = 373$ .

Однією з найпривабливіших властивостей дерев є те, що вони можуть бути графічно представлені. Ми використаємо функцію `plot()` для відображення структури дерева і функцію `text()` для відображення міток вузлів. Аргумент `pretty = 0` вказує R включати назви категорій для будь-яких якісних предикторів, а не просто відображати літери для кожної категорії.

```
> plot(tree.carseats)
> text(tree.carseats, pretty=0)
```

Найважливішим показником, що впливає на Sales виявляється є розташування стелажів, оскільки перша гілка відрізняє Good місця від Bad та Medium.

Якщо ми просто введемо ім'я дерева-об'єкта, R виведе відповідні результати для кожної гілки дерева. R відображає критерій розбиття (наприклад, Price <92,5), кількість спостережень у цій гілці, відхилення, загальний прогноз для гілки (Yes чи No), і частку спостережень у цій гілці які приймають значення Yes чи No. Гілки, що ведуть до термінальних вузлів позначаються зірочками.

```
> tree.carseats
node), split, n, deviance, yval, (yprob)
      * denotes terminal node
 1) root 400 541.5 No ( 0.590 0.410 )
 2) ShelveLoc: Bad,Medium 315 390.6 No ( 0.689 0.311 )
   4) Price < 92.5 46 56.53 Yes ( 0.304 0.696 )
     8) Income < 57 10 12.22 No ( 0.700 0.300 )
```

Для того, щоб правильно оцінити результативність дерева класифікацій на цих даних, ми повинні оцінити тестову помилку, а не просто обчислювати помилку навчання. Ми розділили спостереження на навчальний набір та тестовий, побудували дерево за допомогою навчального набору та оцінили його результативність на тестовому наборі. Для цього може бути використана функція

`predict ()`. У випадку дерева класифікації аргумент `type = "class"` вказує R повернути фактичне передбачення класу. Цей підхід призводить до правильних прогнозів для приблизно 71,5% у наборі тетстових даних.

```
> set.seed(2)
> train=sample(1:nrow(Carseats), 200)
> Carseats.test=Carseats[-train,]
> High.test=High[-train]

> tree.carseats=tree(High~.-Sales,Carseats,subset=train)
> tree.pred=predict(tree.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)

  High.test
tree.pred No Yes
  No    86   27
  Yes   30   57
> (86+57)/200
[1] 0.715
```

Далі ми розглядаємо, чи може обрізка дерева призвести до поліпшення результатів. Функція `cv.tree ()` виконує перехресну перевірку для того, щоб визначити оптимальний рівень складності дерева; `cost complexity pruning` використовується для того, щоб вибрати послідовність дерев для розгляду. Ми використовуємо аргумент `FUN = prune.misclass` для того, щоб вказати, що ми хочемо використати коефіцієнт помилок класифікації для керівництва процесом перехресної перевірки та обрізки, а не девіацію, що використовується за замовчуванням для функції `cv.tree ()`. Функція `cv.tree ()` повідомляє кількість термінальних вузлів кожного дерева, що розглядається (`size`), а також відповідний коефіцієнт помилок і значення параметра складність-витрати, що використовується (`k`).

```

> set.seed(3)
> cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
> names(cv.carseats)
[1] "size"    "dev"      "k"        "method"
> cv.carseats
$size
[1] 19 17 14 13  9  7  3  2  1

$dev
[1] 55 55 53 52 50 56 69 65 80

$k
[1]       -Inf   0.0000000   0.6666667   1.0000000   1.7500000
     2.0000000   4.2500000
[8]   5.0000000 23.0000000

$method
[1] "misclass"

attr(,"class")
[1] "prune"           "tree.sequence"

```

Зверніть увагу, що, незважаючи на називу, dev відповідає помилці перехресної перевірки в цьому випадку. Дерево з 9 кінцевими вузлами призводить до найнижчого коефіцієнта помилок перехресної перевірки з 50 помилками перехресної перевірки. Ми будуємо графік помилки як функція від size і k.

```

> par(mfrow=c(1,2))
> plot(cv.carseats$size ,cv.carseats$dev ,type="b")
> plot(cv.carseats$k ,cv.carseats$dev ,type="b")

```

Тепер ми застосуємо функцію prune.misclass () для того, щоб обрізати дерево та отримати дев'ятивузлове дерево.

```

> prune.carseats=prune.misclass(tree.carseats,best=9)
> plot(prune.carseats)
> text(prune.carseats,pretty=0)

```

Наскільки добре це обрізане дерево працює на тестовому наборі даних Знову, застосуємо функцію predict () .

```

> tree.pred=predict(prune.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
  High.test
tree.pred No Yes
  No    94   24
  Yes   22   60
> (94+60)/200
[1] 0.77

```

Тепер 77% тестових спостережень правильно класифіковано, отже результатом процесу обрізання є не тільки дерево, яке є більш зрозумілим, але й покращена точність класифікації.

Якщо ми збільшимо значення best, ми отримаємо обрізане дерево більшого розміру з нижчою точністю класифікації

```
> prune.carseats=prune.misclass(tree.carseats, best=15)
> plot(prune.carseats)
> text(prune.carseats, pretty=0)
> tree.pred=predict(prune.carseats, Carseats.test, type="class")
> table(tree.pred, High.test)
      High.test
tree.pred No Yes
      No    86   22
      Yes   30   62
> (86+62)/200
[1] 0.74
```

## Дерева регресії

Ми побудуємо дерево регресії для набору даних Boston. Спочатку ми створимо навчальний набір даних та побудуємо дерево на навчальних даних.

```
> library(MASS)
> set.seed(1)
> train = sample(1:nrow(Boston), nrow(Boston)/2)
> tree.boston=tree(medv~., Boston, subset=train)
> summary(tree.boston)

Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "lstat" "rm"     "dis"
Number of terminal nodes:  8

Residual mean deviance:  12.65 = 3099 / 245
Distribution of residuals:
    Min.  1st Qu.  Median    Mean  3rd Qu.  Max. 
-14.1000 -2.0420 -0.0536  0.0000  1.9600 12.6000
```

Результат summary() вказує, що лише три змінні були використані при побудові дерева. В контексті дерева регресії, девіація - це просто сума квадратів помилок для дерева. Зобразимо дерево графічно.

```
> plot(tree.boston)
> text(tree.boston, pretty=0)
```

Змінна lstat вимірює відсоток осіб з нижчим соціально-економічним статусом. Дерево вказує, що нижчі значення lstat відповідають дорожчим будинкам. Дерево передбачає середню ціну будинку 46 400 доларів на більші будинки в районах, в яких мешканці мають високий соціально-економічний статус ( $rm >= 7,437$  і  $lstat < 9,715$ ).

Тепер ми використовуємо функцію cv.tree(), щоб побачити, чи покращить обрізка дерева його результативність.

```
> cv.boston=cv.tree(tree.boston)
> plot(cv.boston$size, cv.boston$dev, type='b')
```

У цьому випадку найбільш складне дерево вибирається шляхом перехресної перевірки. Однак якщо ми хочемо обрізати дерево, ми можемо зробити це наступним чином, використовуючи функції prune.tree():

```
> prune.boston=prune.tree(tree.boston, best=5)
> plot(prune.boston)
> text(prune.boston, pretty=0)
```

Відповідно до результатів перехресної перевірки, ми використовуємо необрізане дерево для побудови прогнозів на тестовому наборі.

```
> yhat=predict(tree.boston, newdata=Boston[-train,])
> boston.test=Boston[-train, "medv"]
> plot(yhat, boston.test)
> abline(0,1)
> mean((yhat-boston.test)^2)
[1] 25.05
```

Іншими словами, тестовий MSE, пов'язаний з деревом регресії, дорівнює 25.05. Отже, квадратний корінь MSE становить близько 5,005, що вказує що ця модель призводить до тестових прогнозів, які лежать в околі \$ 5, 005 від справжнього значення медіани для ціни будинку.

### *Бутстрап агрегація і випадкові ліси*

Застосовуємо бутстррап агрегацію та випадкові ліси до даних Boston, використовуючи randomForest пакет. Результати, отримані в цьому розділі, можуть залежати від версії R та версії пакета randomForest. Нагадаємо, що

бутстррап агрегація - це просто особливий випадок випадкового лісу з  $m = p$ . Отже, функція `randomForest()` може використовуватися для побудови як випадкових лісів, так і бутстррап агрегації. Застосовуємо бутстррап агрегацію наступним чином:

```
> library(randomForest)
> set.seed(1)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,
  mtry=13,importance=TRUE)
> bag.boston

Call:
randomForest(formula = medv ~ ., data = Boston, mtry = 13,
  importance = TRUE, subset = train)
  Type of random forest: regression
  Number of trees: 500
No. of variables tried at each split: 13

  Mean of squared residuals: 10.77
  % Var explained: 86.96
```

Аргумент `mtry = 13` вказує на те, що слід враховувати всі 13 предикторів для кожного поділу дерева - іншими словами, потрібно робити бутстррап агрегацію. Як добре працює ця модель на тестовому наборі?

```
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> plot(yhat.bag, boston.test)
> abline(0,1)
> mean((yhat.bag-boston.test)^2)
[1] 13.16
```

Тестовий MSE, пов'язаний з деревом регресії отриманого на основі бутстррап агрегації, становить майже 13,16, що вдвічі менше, ніж отримано з використанням оптимально обрізаного дерева. Ми могли б змінити кількість дерев, вирощених за допомогою `randomForest()` за допомогою аргументу `ntree`:

```
> bag.boston=randomForest(medv~.,data=Boston,subset=train,
  mtry=13,ntree=25)
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> mean((yhat.bag-boston.test)^2)
[1] 13.31
```

Побудова випадкового лісу відбувається точно так само, за винятком того, що ми використовуємо менше значення аргументу `mtry`. За замовчуванням

`randomForest()` використовує  $p/3$  змінних при побудові дерев регресії, та  $\sqrt{p}$  змінних при побудові дерев класифікації. Використаємо `mtry = 6`.

```
> set.seed(1)
> rf.boston=randomForest(medv~.,data=Boston,subset=train,
+ mtry=6,importance=TRUE)
> yhat.rf = predict(rf.boston,newdata=Boston[-train,])
> mean((yhat.rf-Boston.test)^2)
[1] 11.31
```

Тестовий MSE становить 11,31, що свідчить про те, що випадкові ліси дали покращення порівняно з бутстррап агрегацією в цьому випадку.

Використовуючи функцію `importance()`, можемо дослідити важливість кожної зі змінних.

```
> importance(rf.boston)
      %IncMSE IncNodePurity
crim     12.384      1051.54
zn        2.103       50.31
indus    8.390      1017.64
chas     2.294       56.32
nox      12.791      1107.31
rm       30.754      5917.26
age      10.334      552.27
dis      14.641      1223.93
rad       3.583       84.30
tax       8.139      435.71
ptratio  11.274      817.33
black    8.097      367.00
lstat   30.962      7713.63
```

Наводяться два показники важливості змінних. Перший базується на середньому зниженні точності прогнозування для ОOB вибірок коли дана змінна виключається з моделі. Другий є мірою загального зменшення домішки вузла, що виникає в результаті поділу над цією змінною, усереднена по всіх деревах. У випадку дерев регресії домішка вузла вимірюється тренувальним RSS, а для дерев класифікації за девіацією. Графіки цих показників можуть бути побудовані за допомогою функції `varImpPlot()`.

```
> varImpPlot(rf.boston)
```

Результати показують, що серед всіх деревах, що розглядаються методом випадкового лісу, рівень багатства громади (`lstat`) та розмір будинку (`rm`) є двома найважливішими змінними.

## Підсилення

Використаємо пакет gbm, а в ньому функцію gbm(), щоб побудувати підсилене дерево регресії на наборі даних Boston. Ми запускаємо gbm() з опцією distribution="gaussian", оскільки це дерево регресії; якби це було дерево класифікації, ми б використовували distribution="bernoulli". Аргумент n.trees = 5000 вказує, що ми хочемо побудувати 5000 дерев, а опція interaction.depth = 4 обмежує глибину кожного дерева.

```
> library(gbm)
> set.seed(1)
> boost.boston=gbm(medv~.,data=Boston[train,],distribution=
+ "gaussian",n.trees=5000,interaction.depth=4)
```

Функція summary () буде графік відносного впливу, а також виводить результати статистик відносного впливу.

```
> summary(boost.boston)
      var     rel.inf
1    lstat   45.96
2      rm    31.22
3      dis    6.81
4      crim   4.07
5      nox    2.56
6  ptratio   2.27
7    black   1.80
8      age   1.64
9      tax    1.36
10    indus   1.27
11    chas    0.80
12      rad   0.20
13      zn    0.015
```

Ми бачимо, що lstat і rm є найважливішими змінними. Ми можемо також побудувати графіки часткової залежності для цих двох змінних. Ці графіки ілюструють граничний вплив вибраних змінних на залежну змінну після видалення інших змінних. У цьому випадку, як і можна було очікувати, медіана ціни на житло зростає за rm та спадає за lstat.

```
> par(mfrow=c(1,2))
> plot(boost.boston,i="rm")
> plot(boost.boston,i="lstat")
```

Використаємо підсилену модель для прогнозування medv на тестовому наборі

```
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],  
+ n.trees=5000)  
> mean((yhat.boost-boston.test)^2)  
[1] 11.8
```

Отриманий тестовий MSE становить 11,8 (аналогічно тестовому MSE для випадкових лісів). Якщо ми хочемо, ми можемо виконати підсилення з іншим значенням параметра стиснення  $\lambda$ . За замовчуванням значення дорівнює 0,001, але це легко змінити. Візьмемо  $\lambda = 0,2$ .

```
> boost.boston=gbm(medv~.,data=Boston[train,],distribution=  
+ "gaussian",n.trees=5000,interaction.depth=4,shrinkage=0.2,  
+ verbose=F)  
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],  
+ n.trees=5000)  
> mean((yhat.boost-boston.test)^2)  
[1] 11.5
```

У цьому випадку використання  $\lambda = 0,2$  призводить до трохи нижчого тестового MSE, ніж для  $\lambda = 0,001$ .