

Лабораторна 5: Методи вибору підмножини, гребенева регресія та Лассо, метод головних компонент та часткових найменших квадратів.

Вибір найкращої підмножини

Ми використаємо дані `Hitters` для передбачення зарплати бейсболістів на основі показників за минулий сезон. Перш за все, ми зауважимо, що значення змінної `Salary` відсутнє для деяких з гравців. Функція `is.na()` може бути використана для ідентифікації відсутніх спостережень. Вона повертає вектор тієї ж довжини, що і вхідний вектор, із значенням `TRUE` для будь-яких елементів, які відсутні, і `FALSE` - інакше. Після цього за допомогою функції `sum()` можна обчислити кількість відсутніх елементів.

```
> library(ISLR)
> fix(Hitters)
> names(Hitters)
[1] "AtBat"      "Hits"        "HmRun"       "Runs"        "RBI"
[6] "Walks"       "Years"       "CAtBat"      "CHits"       "CHmRun"
[11] "CRuns"       "CRBI"        "CWalks"      "League"      "Division"
[16] "PutOuts"     "Assists"     "Errors"      "Salary"      "NewLeague"
> dim(Hitters)
[1] 322 20
> sum(is.na(Hitters$Salary))
[1] 59
```

Отже, бачимо, що відсутня інформація про зарплати 59 гравців. Використаємо функцію `na.omit()` для видалення рядків з відсутніми значеннями.

```
> Hitters=na.omit(Hitters)
> dim(Hitters)
[1] 263 20
> sum(is.na(Hitters))
[1] 0
```

Функція `regsubsets()` (у бібліотеці `leaps`) виконує вибір найкращої підмножини шляхом визначення найкращої моделі, що містить задане число предикторів. Якість моделі визначається за допомогою RSS. Синтаксис такий як і для функції

`lm()`. Команда `summary()` виводить найкращий набір змінних для кожної розмірності моделі.

```
> library(leaps)
> regfit.full=regsubsets(Salary~., Hitters)
> summary(regfit.full)
Subset selection object
Call: regsubsets.formula(Salary ~ ., Hitters)
19 Variables (and intercept)
...
1 subsets of each size up to 8
Selection Algorithm: exhaustive
      AtBat  Hits  HmRun  Runs  RBI  Walks  Years  CAtBat  CHits
1 ( 1 ) " "   " "   " "   " "   " "   " "   " "   " "
2 ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "
3 ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "
4 ( 1 ) " "   "*"  " "   " "   " "   " "   " "   " "
5 ( 1 ) "*"  "*"  " "   " "   " "   " "   " "   " "
6 ( 1 ) "*"  "*"  " "   " "   " "   "*"  " "   " "
7 ( 1 ) " "   "*"  " "   " "   " "   "*"  " "   "*"
8 ( 1 ) "*"  "*"  " "   " "   " "   "*"  " "   " "
      CHmRun  CRuns  CRBI  CWalks  LeagueN  DivisionW  PutOuts
1 ( 1 ) " "   " "   "*"  " "   " "   " "   " "
2 ( 1 ) " "   " "   "*"  " "   " "   " "   " "
3 ( 1 ) " "   " "   "*"  " "   " "   " "   "*"
4 ( 1 ) " "   " "   "*"  " "   " "   "*"  "*"
5 ( 1 ) " "   " "   "*"  " "   " "   "*"  "*"
6 ( 1 ) " "   " "   "*"  " "   " "   "*"  "*"
7 ( 1 ) "*"  " "   " "   " "   " "   "*"  "*"
8 ( 1 ) "*"  "*"  " "   "*"  " "   "*"  "*"
      Assists  Errors  NewLeagueN
1 ( 1 ) " "   " "   " "
2 ( 1 ) " "   " "   " "
3 ( 1 ) " "   " "   " "
4 ( 1 ) " "   " "   " "
5 ( 1 ) " "   " "   " "
6 ( 1 ) " "   " "   " "
7 ( 1 ) " "   " "   " "
8 ( 1 ) " "   " "   " "
```

Зірочка вказує, що дана змінна включена у відповідну модель. Наприклад, цей результат вказує, що найкраща модель із двома змінними містить лише `Heats` та `CRBI`. За замовчуванням `regsubsets()` повідомляє лише про результати до найкращої моделі з восьми змінних. Але можна використати параметр `nmax`, щоб повернути скільки завгодно змінних. Тут ми розглянемо моделі з кількістю змінних до 19 включно.

```
> regfit.full=regsubsets(Salary~., data=Hitters, nvmax=19)
> reg.summary=summary(regfit.full)
```

Функція `summary()` повертає також значення R^2 , скорегований R^2 , C_p , BIC , RSS . Використаємо це для ідентифікації найкращої моделі загалом.

```
> names(reg.summary)
[1] "which"    "rsq"      "rss"       "adjr2"     "cp"        "bic"
[7] "outmat"   "obj"
```

Наприклад, ми бачимо, що статистика R^2 зростає з 32%, коли в модель включена лише одна змінна до майже до 55%, коли всі змінні включені. Як і очікувалось, статистика R^2 зростає монотонно, з кількістю включених змінних.

```
> reg.summary$rsq
[1] 0.321 0.425 0.451 0.475 0.491 0.509 0.514 0.529 0.535
[10] 0.540 0.543 0.544 0.544 0.545 0.545 0.546 0.546 0.546
[19] 0.546
```

Зобразимо графічно R^2 , скорегований R^2 , C_p , BIC , RSS для всіх моделей, щоб спростити процедуру вибору найкращої моделі. Для побудови неперервних графіків використаємо опцію `type = "l"`.

```
> par(mfrow=c(2,2))
> plot(reg.summary$rss, xlab="Number of Variables", ylab="RSS",
       type="l")
> plot(reg.summary$adjr2, xlab="Number of Variables",
       ylab="Adjusted RSq", type="l")
```

Команда `points()` працює як команда `plot()`, за винятком того, що вона додає точки на вже створений графік, замість того, щоб створювати новий. Функцію `which.max()` можна використати для ідентифікації розташування максимуму. Ми намалюємо червону точку для позначення моделі із найбільшим скорегованим R^2 .

```
> which.max(reg.summary$adjr2)
[1] 11
> points(11, reg.summary$adjr2[11], col="red", cex=2, pch=20)
```

Аналогічно зобразимо показники C_p , BIC та за допомогою функції `which.min()` позначимо модель з мінімальними значеннями.

```

> plot(reg.summary$cp, xlab="Number of Variables", ylab="Cp",
      type='l')
> which.min(reg.summary$cp)
[1] 10
> points(10, reg.summary$cp [10], col="red", cex=2, pch=20)
> which.min(reg.summary$bic)
[1] 6
> plot(reg.summary$bic, xlab="Number of Variables", ylab="BIC",
      type='l')
> points(6, reg.summary$bic [6], col="red", cex=2, pch=20)

```

Функція `regsubsets()` має вбудовану команду `plot()`, яка може використовуватися для відображення вибраних змінних для найкращої моделі із заданою кількістю предикторів, ранжованих відповідно до C_p , BIC , AIC , скорегованого R^2 .

```

> plot(regfit.full, scale="r2")
> plot(regfit.full, scale="adjr2")
> plot(regfit.full, scale="Cp")
> plot(regfit.full, scale="bic")

```

Верхній рядок кожного графіка містить чорний квадрат для кожної вибраної змінної відповідно до оптимальної моделі, пов'язаної з цією статистикою. Наприклад, ми бачимо, що декілька моделей мають BIC , близький до -150. Однак модель з найнижчим BIC - це модель із шістьма змінними, яка містить лише AtBat, Hits, Walks, CRBI, DivisionW та PutOuts. Ми можемо використати функцію `coef()`, щоб побачити оцінки коефіцієнтів цієї моделі.

```

> coef(regfit.full, 6)
(Intercept)      AtBat       Hits       Walks       CRBI
               91.512     -1.869      7.604      3.698      0.643
DivisionW      PutOuts
              -122.952      0.264

```

Покроковий вибір вперед та назад

З допомогою функції `regsubsets()` можна вибрати найкращу модель на основі покрокового вибору вперед та назад задавши опцію `method="forward"` чи `method="backward"`.

```

> regfit.fwd=regsubsets(Salary~., data=Hitters ,nvmax=19 ,
  method="forward")
> summary (regfit.fwd)
> regfit.bwd=regsubsets(Salary~., data=Hitters ,nvmax=19 ,
  method="backward ")
> summary (regfit.bwd)

```

Наприклад, ми бачимо, що на основі покрокового відбору вперед отримали, що найкраща модель з однією змінною містить CRBI, а найкраща модель із двома змінними додатково включає Hits. Для цих даних найкращі моделі з однією, двома, ..., шістьма змінними є однакові для методів найкращої підможини та прямого вибору. Однак найкращі моделі із семи змінних є різними для всіх трьох методів відбору.

```

> coef(regfit.full,7)
(Intercept)      Hits       Walks      CAtBat      CHits
  79.451       1.283      3.227     -0.375      1.496
  CHmRun      DivisionW      PutOuts
  1.442      -129.987      0.237
> coef(regfit.fwd,7)
(Intercept)      AtBat      Hits       Walks      CRBI
  109.787      -1.959      7.450      4.913      0.854
  CWalks      DivisionW      PutOuts
  -0.305      -127.122      0.253
> coef(regfit.bwd,7)
(Intercept)      AtBat      Hits       Walks      CRuns
  105.649      -1.976      6.757      6.056      1.129
  CWalks      DivisionW      PutOuts
  -0.716      -116.169      0.303

```

Вибір моделі на основі методів валідаційної множини та перехресної перевірки

Ми продемонстрували вибір найкращої моделі на основі показників C_p , BIC , та скорегованого R^2 . Розглянемо, як це можна зробити з використання методів валідаційної множини та перехресної перевірки. Для того, щоб ці підходи давали правильні оцінки тестової помилки потрібно використовувати лише навчальні спостереження для виконання всіх аспектів підбору моделі - включаючи вибір змінних. Тобто визначення, яка модель даного розміру є найкраща, повинно бути зроблено, використовуючи лише навчальні спостереження. Якщо використовувати повний набір даних для вибору найкращої підмножини, то

помилки валідаційної множини та перехресної перевірки, не будуть правильними оцінками тестової помилки.

Для того, щоб використати підхід валідаційної множини, ми почнемо з розділення всього набору спостережень на навчальний набір та тестовий набір. Ми робимо це, створюючи випадковий вектор, `train`, елементів, що дорівнюють `TRUE`, якщо відповідне спостереження знаходиться в навчальному наборі, і `FALSE` в іншому випадку. Елементи вектора `test` навпаки приймають значення `TRUE`, якщо спостереження знаходиться в тестовому наборі, і `FALSE` - інакше.

```
> set.seed(1)
> train=sample(c(TRUE, FALSE), nrow(Hitters), rep=TRUE)
> test=!train
```

Застосуємо функцію `regsubsets()` до навчального набору, щоб провести вибір найкращої підмножини.

```
> regfit.best=regsubsets(Salary~., data=Hitters[train,],
  nvmax=19)
```

Зверніть увагу, що ми виділяємо підмножину з даних `Hitters` безпосередньо у виклику функції, щоб отримати доступ лише до навчальної підмножини даних, використовуючи вираз `Hitters[train,]`. Обчислимо помилку валідаційної множини для найкращої моделі кожної розмірності. Спочатку створимо модельну матрицю з тестових даних.

```
test.mat=model.matrix(Salary~., data=Hitters [test,])
```

Функція `model.matrix()` використовується у багатьох пакунках регресії для створення матриці "X" на основі даних. Тепер ми запускаємо цикл, і дляожної розмірності моделі i , витягуємо коефіцієнти з `regfit.best` для найкращої моделі такої розмірності, множимо їх на відповідні стовпці тестової матриці моделі для обчислення прогнозу та обчислюємо тестовий MSE.

```
> val.errors=rep(NA, 19)
> for(i in 1:19){
+   coefi=coef(regfit.best, id=i)
+   pred=test.mat[, names(coefi)]%*%coefi
+   val.errors[i]=mean((Hitters$Salary [test]-pred)^2)
}
```

Отримаємо, що найкращою є модель з 10 змінними.

```

> val.errors
[1] 220968 169157 178518 163426 168418 171271 162377 157909
[9] 154056 148162 151156 151742 152214 157359 158541 158743
[17] 159973 159860 160106
> which.min(val.errors)
[1] 10
> coef(regfit.best,10)
(Intercept) AtBat Hits Walks CAtBat
-80.275 -1.468 7.163 3.643 -0.186
CHits CHmRun CWalks LeagueN DivisionW
1.105 1.384 -0.748 84.558 -53.029
PutOuts
0.238

```

Оскільки для функції regsubsets() не передбачено методу predict, то можемо написати свою функцію

```

> predict.regsubsets=function (object,newdata,id){
+ form=as.formula (object$call [[2]])
+ mat=model.matrix (form,newdata )
+ coefi=coef(object,id=id)
+ xvars=names (coefi )
+ mat[,xvars ]%*% coefi
+ }

```

Нарешті, ми виконаємо вибір найкращої підмножини на повному наборі даних і вибираємо найкращу модель із десятьма змінними. Ми використовємо повний набір даних для отримання більш точних оцінок коефіцієнтів. Ми виконуємо вибір найкращої підмножини на повному наборі даних і вибираємо найкращу модель з десятьма змінними, а не просто використовуємо отримані змінні з навчального набору, оскільки найкраща модель із десятьма змінними за повними даними може відрізнятися від відповідної моделі отриманої на навчальному наборі.

```

> regfit.best=regsubsets(Salary~.,data=Hitters ,nvmax=19)
> coef(regfit.best,10)
(Intercept)          AtBat          Hits          Walks          CAtBat
  162.535        -2.169         6.918         5.773        -0.130
   CRuns          CRBI          CWalks      DivisionW          PutOuts
   1.408         0.774        -0.831       -112.380         0.297
Assists
  0.283

```

Насправді ми бачимо, що найкраща модель із десятма змінними на повному наборі даних має інший набір змінних, ніж найкраща модель із десятма змінними на навчальних даних.

Продемонструємо вибір найкращої моделі, використовуючи перехресну перевірку. Цей підхід дещо заплутаний, оскільки ми маємо виконувати вибір найкращої підмножини в кожному з k навчальних наборів. Ми створимо вектор, який відносить кожне спостереження в одну з $k = 10$ множин і матрицю, в якій будемо зберігати результати.

```

> k=10
> set.seed(1)
> folds=sample(1:k,nrow(Hitters),replace=TRUE)
> cv.errors=matrix(NA,k,19, dimnames=list(NULL, paste(1:19)))

```

Напишемо цикл `for`, який виконує перехресну перевірку. У j -му проході елементи `folds`, що дорівнюють j , знаходяться в тестовому наборі, а решта знаходяться в навчальному. Ми будуємо прогнози для моделі кожної розмірності, обчислюємо тестові помилки для відповідної підмножини, і зберігаємо їх у матрицю `cv.errors`.

```

> for(j in 1:k){
+ best.fit=regsubsets(Salary~.,data=Hitters [folds !=j],
+ nvmax=19)
+ for(i in 1:19) {
+ pred=predict.regsubsets(best.fit,Hitters [folds ==j], id=i)
+ cv.errors [j,i]=mean( (Hitters$Salary[folds ==j]-pred)^2)
+ }
+ }

```

Ми отримали матрицю 10×19 , (i, j) -ий елемент якої відповідає тестовому MSE для i -ї перехресної перевірки кращої j -змінної моделі. З допомогою функції `apply()`

усереднемо значення по стовпцях матриці, щоб отримати вектор, для якого j -й елемент є помилкою перехресної перевірки для моделі з j змінними.

```
> mean.cv.errors=apply(cv.errors,2,mean)
> mean.cv.errors
[1] 160093 140197 153117 151159 146841 138303 144346 130208
[9] 129460 125335 125154 128274 133461 133975 131826 131883
[17] 132751 133096 132805
> par(mfrow=c(1,1))
> plot(mean.cv.errors,type='b')
```

Ми бачимо, що перехресна перевірка вибирає модель із 11 змінними. Проведемо вибір найкращої підмножини на повному наборі даних для отримання 11-змінної моделі.

```
> reg.best=regsubsets(Salary~.,data=Hitters, nvmax=19)
> coef(reg.best,11)
(Intercept) AtBat Hits Walks CAtBat
135.751 -2.128 6.924 5.620 -0.139
CRuns CRBI CWalks LeagueN DivisionW
1.455 0.785 -0.823 43.112 -111.146
PutOuts Assists
0.289 0.269
```

Гребенева регресія та ласо

Ми використаємо пакет `glmnet` для того, щоб здійснити гребеневу регресію і ласо. Основною функцією цього пакету є `glmnet()`, яку можна використовувати для побудови гребеневої регресії, ласо тощо. Ця функція має дещо інший синтаксис у порівнянні з функціями, які ми використовували раніше. Зокрема, ми повинні передати матрицю x та вектор y , і не використовується синтаксис $\sim x$. Ми виконаємо гребеневу регресію та ласо, щоб передбачити `Salary` на даних `Hitters`.

```
> x=model.matrix(Salary~.,Hitters)[,-1]
> y=Hitters$Salary
```

Функція `model.matrix()` особливо корисна для створення x ; вона не тільки створює матрицю, що відповідає 19 предикторам, але вона також автоматично перетворює будь-які якісні змінні в фіктивні. Остання властивість важлива, оскільки `glmnet()` може приймати лише числові, кількісні вхідні дані.

Гребенева регресія

Функція `glmnet()` має аргумент `alpha`, який визначає, який тип моделі розглядається. Якщо $\alpha = 0$, то гребенева регресія, а якщо $\alpha = 1$ то ласо. Розглянемо спочатку гребеневу регресію.

```
> library(glmnet)
> grid=10^seq(10,-2,length=100)
> ridge.mod=glmnet(x,y,alpha=0,lambda=grid)
```

За замовчуванням функція `glmnet()` автоматично використовує гребеневу регресію для автоматично вибраного діапазону значень λ . Ми реалізуємо функції над значеннями від $\lambda = 10^{10}$ до $\lambda = 10^{-2}$, які по суті охоплюють весь спектр сценаріїв. Ми також можемо будувати модель для певного значення λ . Зверніть увагу, що за замовчуванням функція `glmnet()` стандартизує змінні, щоб вони були в одному масштабі. Щоб вимкнути це налаштування за замовчуванням, використайте аргумент `standardize = FALSE`. З кожним значенням λ пов'язаний вектор коефіцієнтів гребеневої регресії, які зберігаються в матриці, до якої можна отримати доступ через `coef()`. У цьому випадку це 20×100 матриця, з 20 рядками та 100 стовпцями (по одному для кожного значення λ).

```
> dim(coef(ridge.mod))
[1] 20 100
```

Ми очікуємо, що оцінки коефіцієнтів будуть набагато менші, з точки зору l_2 норми, коли використовується велике значення λ . Наведемо коефіцієнти, коли $\lambda = 11498$, разом з їхніми l_2 нормами

```
> ridge.mod$lambda[50]
[1] 11498
> coef(ridge.mod)[,50]
(Intercept) AtBat Hits HmRun Runs
407.356 0.037 0.138 0.525 0.231
RBI Walks Years CAtBat CHits
0.240 0.290 1.108 0.003 0.012
CHmRun CRuns CRBI CWalks LeagueN
0.088 0.023 0.024 0.025 0.085
DivisionW PutOuts Assists Errors NewLeagueN
-6.215 0.016 0.003 -0.021 0.301
> sqrt(sum(coef(ridge.mod)[-1,50]^2))
[1] 6.36
```

При $\lambda = 705$, отримаємо

```

> ridge.mod$lambda[60]
[1] 705
> coef(ridge.mod)[,60]
(Intercept) AtBat Hits HmRun Runs
      54.325   0.112 0.656 1.180 0.938
      RBI     Walks Years CAtBat CHits
      0.847   1.320 2.596 0.011 0.047
      CHmRun CRuns CRBI CWalks LeagueN
      0.338   0.094 0.098 0.072 13.684
      DivisionW PutOuts Assists Errors NewLeagueN
      -54.659   0.119 0.016 -0.704 8.612
> sqrt(sum(coef(ridge.mod)[-1,60]^2))
[1] 57.1

```

Ми можемо використати функцію predict() для ряду цілей. Наприклад, ми можемо отримати коефіцієнти гребеневої регресії для нового значення λ , скажімо 50

```

> predict(ridge.mod,s=50,type="coefficients")[1:20,]
(Intercept) AtBat Hits HmRun Runs
      48.766  -0.358 1.969 -1.278 1.146
      RBI     Walks Years CAtBat CHits
      0.804   2.716 -6.218 0.005 0.106
      CHmRun CRuns CRBI CWalks LeagueN
      0.624   0.221 0.219 -0.150 45.926
      DivisionW PutOuts Assists Errors NewLeagueN
      -118.201   0.250 0.122 -3.279 -9.497

```

Розділимо вибірку на навчальний та тестовий набори для оцінки тестової помилки. Є два типові способи випадкового поділу набору даних. Перший - це зробити випадковий вектор TRUE, FALSE елементів і вибирати спостереження, що відповідають значенню TRUE для навчальних даних. Другий - це випадковий вибір підмножини чисел від 1 до n; їх потім можна використовувати як індекси для навчальних спостережень. Використаємо другий підхід.

```

> set.seed(1)
> train=sample(1:nrow(x), nrow(x)/2)
> test=(-train)
> y.test=y[test]

```

Ми оцінюємо гребеневу регресію на навчальному наборі та проводимо оцінку MSE на тестовому наборі, використовуючи $\lambda = 4$. Ми використовуємо функцію predict() для побудови прогнозів для тестового набору, замінивши type="coefficients" на аргумент newx.

```

> ridge.mod=glmnet(x[train,],y[train],alpha=0,lambda=grid,
  thresh=1e-12)
> ridge.pred=predict(ridge.mod,s=4,newx=x[test,])
> mean((ridge.pred-y.test)^2)
[1] 101037

```

Тестове MSE - 101037. Якби ми розглянули модель без предикторів (всі коефіцієнти = 0), ми б передбачили кожне тестове спостереження, використовуючи середнє значення навчальних спостережень. У цьому випадку тестове MSE

```

> mean((mean(y[train])-y.test)^2)
[1] 193253

```

Ми також могли б отримати той самий результат, використавши гребеневу регресію з дуже великим значенням λ .

```

> ridge.pred=predict(ridge.mod,s=1e10,newx=x[test,])
> mean((ridge.pred-y.test)^2)
[1] 193253

```

Перевіримо, чи гребенева регресія з $\lambda = 4$ краща ніж звичайна (регресія оцінена методом найменших квадратів).

```

> ridge.pred=predict(ridge.mod,s=0,newx=x[test,],exact=T)
> mean((ridge.pred-y.test)^2)
[1] 114783
> lm(y~x, subset=train)
> predict(ridge.mod,s=0,exact=T,type="coefficients") [1:20,]

```

Загалом, якщо ми хочемо використати метод найменших квадратів, тоді ми повинні використовувати функцію `lm()`, оскільки ця функція надає більше інформації про результати, такої як стандартні помилки та p-значення для коефіцієнтів. Загалом, замість довільного вибору $\lambda = 4$, було б краще використати перехресну перевірку, щоб вибрати значення параметра λ . Ми можемо зробити це за допомогою вбудованої функція перехресної перевірки, `cv.glmnet()`. За замовчуванням функція `cv.glmnet()` виконує десятикратну перехресну перевірку, хоча це можна змінити за допомогою аргумента `folds`.

```

> set.seed(1)
> cv.out=cv.glmnet(x[train,],y[train],alpha=0)
> plot(cv.out)
> bestlam=cv.out$lambda.min
> bestlam
[1] 212

```

Отже, значення λ , яке призводить до найменшої помилки перехресної перевірки дорівнює 212. Як значення тестового MSE, пов'язаний з цим значенням λ ?

```
> ridge.pred=predict(ridge.mod,s=bestlam,newx=x[test,])
> mean((ridge.pred-y.test)^2)
[1] 96016
```

Тобто ми отримали покращення порівняно з тестовим MSE, який ми отримали для $\lambda = 4$. Нарешті, ми оцінили модель гребеневої регресії на повному наборі даних, використовуючи значення λ , вибране перехресною перевіркою, і дослідили оцінки коефіцієнтів.

```
> out=glmnet(x,y,alpha=0)
> predict(out,type="coefficients",s=bestlam)[1:20,]
(Intercept) AtBat Hits HmRun Runs
 9.8849    0.0314 1.0088 0.1393 1.1132
   RBI      Walks Years CAtBat CHits
 0.8732    1.8041 0.1307 0.0111 0.0649
   CHmRun   CRuns CRBI CWalks LeagueN
 0.4516    0.1290 0.1374 0.0291 27.1823
DivisionW PutOuts Assists Errors NewLeagueN
 -91.6341   0.1915 0.0425 -1.8124 7.2121
```

Як і слід було очікувати, жоден з коефіцієнтів не дорівнює нулю - гребенева регресія не виконує вибір змінних!

Ласо

Ми побачили, що гребенева регресія з розумним вибором λ може бути кращою ніж регресія найменших квадратів та нульова модель на наборі даних Hitters. Виникає питання, чи може ласо дати точніші або зрозуміліші результати, ніж гребенева регресія. Для того, щоб підібрати модель ласо, ми ще раз використаємо функцію `glmnet()`; однак цього разу ми використаємо аргумент `alpha = 1`. Все решта так само, як і при побудові моделі гребеневої регресії.

```
> lasso.mod=glmnet(x[train,],y[train],alpha=1,lambda=grid)
> plot(lasso.mod)
```

З графіку коефіцієнтів бачимо, що залежно від вибору параметра, деякі коефіцієнти будуть дорівнювати нулю. Ми виконаємо перехресну перевірку та обчислимо відповідні тестові помилки.

```

> set.seed(1)
> cv.out=cv.glmnet(x[train,],y[train],alpha=1)
> plot(cv.out)
> bestlam=cv.out$lambda.min
> lasso.pred=predict(lasso.mod,s=bestlam,newx=x[test,])
> mean((lasso.pred-y.test)^2)
[1] 100743

```

Отримали значення значно нижче, ніж для нульової моделі та методу найменших квадратів, і дуже схоже на гребеневу регресію з λ обраним шляхом перехресної перевірки. Однак ласо має суттєву перевагу перед гребеневою регресією, оскільки оцінки 12 з 19 коефіцієнтів дорівнюють нулю. Отже, модель ласо з λ обраним перехресною перевіркою містить лише сім змінних.

```

> out=glmnet(x,y,alpha=1,lambda=grid)
> lasso.coef=predict(out,type="coefficients",s=bestlam)[1:20,]
> lasso.coef
(Intercept) AtBat Hits HmRun Runs
18.539 0.000 1.874 0.000 0.000
RBI Walks Years CAtBat CHits
0.000 2.218 0.000 0.000 0.000
CChmRun CRuns CRBI CWalks LeagueN
0.000 0.207 0.413 0.000 3.267
DivisionW PutOuts Assists Errors NewLeagueN
-103.485 0.220 0.000 0.000 0.000
> lasso.coef[lasso.coef!=0]
(Intercept) Hits Walks CRuns CRBI
18.539 1.874 2.218 0.207 0.413
LeagueN DivisionW PutOuts
3.267 -103.485 0.220

```

Метод головних компонент (PCR)

Метод головних компонент може бути виконаний за допомогою функції pcr(), яка є частиною бібліотеки pls. Ми застосовуємо PCR до даних Hitters, щоб передбачити Salary.

```

> library(pls)
> set.seed(2)
> pcr.fit=pcr(Salary~., data=Hitters, scale=TRUE,
  validation="CV")

```

Синтаксис функції pcr() подібний до синтаксису функції lm(), з деякими додатковими опціями. Встановлення scale=TRUE має наслідком стандартизацію кожного предиктора, щоб шкала, за якою вимірюється кожна змінна, не мала

істотного впливу. Налаштування validation="CV" змушує pcr() обчислювати помилку перехресної перевірки ($k=10$) для кожного можливого значення M , кількості використаних головних компонентів. Результати отримуємо за допомогою summary().

```
> summary(pcr.fit)
Data: X dimension: 263 19
      Y dimension: 263 1
Fit method: svdpc
Number of components considered: 19

VALIDATION: RMSEP
Cross-validated using 10 random segments.
              (Intercept) 1 comps  2 comps  3 comps  4 comps
CV            452       348.9    352.2    353.5    352.8
adjCV         452       348.7    351.8    352.9    352.1
...
              ...
TRAINING: % variance explained
          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
X            38.31    60.16    70.84    79.03    84.29    88.63
Salary        40.63    41.58    42.17    43.22    44.90    46.48
...
```

Оцінка CV надається дляожної можливої кількості компонентів, починаючи з $M = 0$. Зверніть увагу, що pcr() повертає корінь з MSE; для того, щоб отримати звичайне значення MSE, ми повинні піднести це значення до квадрату.

Також можна побудувати графік оцінок перехресної перевірки, використовуючи функцію validationplot(). Використання val.type="MSEP" призведе до того, що MSE перехресної перевірки буде представлено графічно.

```
> validationplot(pcr.fit, val.type="MSEP")
```

Ми бачимо, що найменша помилка перехресної перевірки виникає, коли $M = 16$. Це не набагато менше, ніж $M = 19$. Однак із графіку ми бачимо, що помилка перехресної перевірки приблизно однакова, навіть коли лише одна компонента включена в модель. Це говорить про те, що можливо достатньо використати модель з невеликою кількістю компонент.

Функція summary() надає відсоток поясненої дисперсії з використанням різної кількості компонентів. Коротко, ми можемо сприймати це як кількість інформації про предиктори або залежну змінну, яка фіксується за допомогою M основних компонентів. Наприклад, встановлення $M = 1$ фіксує лише 38,31% всієї

дисперсії або інформації в предикторах. На відміну від цього, використання $M = 6$ збільшує значення до 88,63%. Якщо ми б використали всі $M = p = 19$ компонентів, це значення зросло б до 100%. Проведемо PCR на навчальних даних та оцінимо його поведінку на тестовому наборі.

```
> set.seed(1)
> pcr.fit=pcr(Salary~., data=Hitters, subset=train, scale=TRUE,
   validation="CV")
> validationplot(pcr.fit, val.type="MSEP")
```

Виявляється, що найменша помилка перехресної перевірки виникає, коли $M = 7$. Ми обчислюємо тестовий MSE наступним чином.

```
> pcr.pred=predict(pcr.fit, x[test,], ncomp=7)
> mean((pcr.pred-y.test)^2)
[1] 96556
```

Цей тестовий MSE є порівняним з результатами, отриманими за допомогою гребеневої регресії і ласо. Однак, як результат використання PCR, остаточну модель складніше інтерпретувати.

Нарешті, ми застосуємо PCR на повному наборі даних, використовуючи $M = 7$, кількість компонент, ідентифікована шляхом перехресної перевірки.

```
> pcr.fit=pcr(y~x, scale=TRUE, ncomp=7)
> summary(pcr.fit)
Data: X dimension: 263 19
      Y dimension: 263 1
Fit method: svdpc
Number of components considered: 7
TRAINING: % variance explained
    1 comps   2 comps   3 comps   4 comps   5 comps   6 comps
X 38.31     60.16    70.84    79.03    84.29    88.63
y 40.63     41.58    42.17    43.22    44.90    46.48
    7 comps
X 92.26
y 46.69
```

Метод часткових найменших квадратів (PLS)

Метод часткових найменших квадратів (PLS) реалізується функцією plsr() з бібліотеки pls. Синтаксис подібний до синтаксису функції pcr().

```

> set.seed(1)
> pls.fit=plsrf(Salary~., data=Hitters, subset=train, scale=TRUE,
+ validation="CV")
> summary(pls.fit)
Data: X dimension: 131 19
      Y dimension: 131 1
Fit method: kernelpls
Number of components considered: 19

VALIDATION: RMSEP
Cross-validated using 10 random segments.
          (Intercept) 1 comps 2 comps 3 comps 4 comps
CV          464.6    394.2    391.5    393.1    395.0
adjCV       464.6    393.4    390.2    391.1    392.9
...
TRAINING: % variance explained
          1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
X          38.12   53.46   66.05   74.49   79.33   84.56
Salary     33.58   38.96   41.57   42.43   44.04   45.59
...
> validationplot(pls.fit, val.type="MSEP")

```

Найменша помилка перехресної перевірки виникає, коли $M = 2$. Оцінимо відповідний тестовий MSE.

```

> pls.pred=predict(pls.fit, x[test,], ncomp=2)
> mean((pls.pred-y.test)^2)
[1] 101417

```

Тестовий MSE порівнянний з тестовими MSE гребеневої регресії, ласо та PCR, але трохи вищий за них. Нарешті, ми застосуємо PLS на повному наборі даних, використовуючи $M = 2$ компонентів, ідентифікованих шляхом перехресної перевірки.

```

> pls.fit=plsrf(Salary~., data=Hitters, scale=TRUE, ncomp=2)
> summary(pls.fit)
Data: X dimension: 263 19
      Y dimension: 263 1
Fit method: kernelpls
Number of components considered: 2
TRAINING: % variance explained
          1 comps 2 comps
X          38.08   51.03
Salary     43.05   46.40

```

Зверніть увагу, що відсоток поясненої дисперсії в Salary складає 46,40%, що майже стільки ж, скільки пояснюється використанням семикомпонентної моделі

PCR, 46,69%. Це тому, що PCR максимізує величину дисперсії, пояснену в предикторах, натомість PLS шукає напрямки, що пояснюють дисперсію в предикторах і у залежній змінній.