Лекція 14. Пакет caret

У цій лекції ви зосередитеся на пакеті caret, інструментах для створення властивостей і попередній обробці даних.

Мета заняття:

- використовувати пакет caret для ефективної побудови та оцінки моделі;
- запровадити методи нарізки даних для ефективного розділення наборів даних для навчання та тестування;
- вивчити різні варіанти навчання, доступні для підгонки та оптимізації моделі.

Зміст:

1. Пакет caret

Пакет caret – це дуже корисний пакет, який містить багато алгоритмів прогнозування та інструментів, які ви використовуватимете в мові програмування R. Функціональні можливості, вбудовані в пакет caret, є одними з наступних. Наприклад, ви можете використовувати інструменти попередньої обробки в пакеті caret, щоб очистити дані та налаштувати властивості, щоб їх можна було використовувати для прогнозування. За допомогою функцій createDataPartition і createTimeSlices можна також виконувати перехресну перевірку та розділення даних у навчальному наборі. Ви також можете створювати навчальні та тестові набори за допомогою функцій train та predict. Ви також можете виконати порівняння моделі за допомогою функції confusionMatrix, яка надасть вам інформацію про те, наскільки добре модель працює на нових наборах даних.

У R доступні численні алгоритми машинного навчання, починаючи від широко використовуваних статистичних методів, таких як лінійний дискримінантний аналіз і регресія, до більш поширених в інформатиці, таких як метод опорних векторів, дерева класифікації та регресії, випадкові ліси та підсилення. Ці алгоритми розроблені

різними розробниками з різним досвідом, що призводить до дещо різних інтерфейсів для кожного алгоритму прогнозування.

obj Class	Package	predict Function Syntax
lda	MASS	predict(obj) (no options needed)
glm	stats	predict(obj, type = "response")
gbm	gbm	predict(obj, type = "response", n.trees)
mda	mda	predict(obj, type = "posterior")
rpart	rpart	predict(obj, type = "prob")
Weka	RWeka	predict(obj, type = "probability")
LogitBoost	caTools	predict(obj, type = "raw", nIter)

Пакет caret пропонує уніфіковану структуру, яка дозволяє прогнозувати за допомогою однієї функції, усуваючи необхідність вказувати всі відповідні параметри для отримання бажаного прогнозу.

```
library(caret)
library(kernlab)
data(spam)
inTrain <- createDataPartition(y=spam$type, p=0.75, list=FALSE)
training <- spam[inTrain,]
testing <- spam[-inTrain,]
dim(training)
# [1] 3451   58
```

Ось короткий приклад використання пакету caret. Деталі цього процесу будуть розглянуті більш конкретно в наступних прикладах. Спочатку завантажується пакет caret разом із пакетом kernlab для доступу до набору даних spam. Потім дані розділяються на навчальний і тестовий набір. При використанні spam$type приблизно 75% даних виділяється для навчання моделі, а 25% – для тестування. Згодом дані розміщуються в набір навчальних даних за допомогою об’єкта inTrain, створеного з createDataPartition. Подібним чином набір даних тестування створюється шляхом ідентифікації всіх зразків, які не входять до навчального набору. Це призводить до підмножини даних виключно для навчання та іншої підмножини виключно для тестування, і все це досягається через простий інтерфейс.

```
set.seed(32343)
modelFit <- train(type ~., data=training, method="glm")
modelFit
# Generalized Linear Model
#
# 3451 samples
#   57 predictors
#    2 classes: 'nonspam', 'spam'
#
# No pre-processing
# Resampling: Bootstrapped (25 reps)
#
# Summary of sample sizes: 3451, 3451, 3451, 3451, 3451, 3451, …
#
# Resampling results
```

```
#
#   Accuracy  Kappa  Accuracy SD  Kappa SD
#   0.9       0.8    0.02         0.04
```

Далі можна підганяти модель. Для цього використовується команда train з пакету caret. Метою залишається прогнозувати змінну type. Символ тильди, після якого йде крапка, вказує, що всі інші змінні у фреймі даних використовуються для прогнозування type. Навчальний набір даних, створений на попередньому кроці, вказується для побудови моделі. Потім вказуються бажані методи підгонки моделі, наприклад “glm” або інші доступні моделі. Функція train генерує підгонку моделі, використовуючи вказаний набір даних і предиктори. У цьому прикладі модель навчається з використанням 3451 зразка та 57 предикторів для прогнозування членства в класі на основі моделі GLM. Використовуються різні методи тестування продуктивності моделі та вибору найкращої моделі. Для оцінки ефективності моделі та виправлення можливих зміщень, спричинених початковою вибіркою, використовуються методи повторної вибірки, включаючи бутстрепінг з 25 повторами.

```
modelFit$finalModel
# Call:  NULL
#
# Coefficients:
#       (Intercept)               make            address                all
#         -1.490e+00         -3.347e-01         -1.608e-01           1.106e-01
#             num3d                our               over             remove
#          2.570e+00          4.723e-01          1.133e+00           2.881e+00
#          internet              order               mail            receive
# ...
#   charExclamation         charDollar           charHash         capitalAve
#          2.354e-01          4.809e+00          2.703e+00          1.019e-02
#       capitalLong       capitalTotal
#          9.531e-03          9.182e-04
#
# Degrees of Freedom: 3450 Total (i.e. Null);  3393 Residual
# Null Deviance:        4628
# Residual Deviance: 1321 AIC: 1437
```

Коли модель підігнана, її можна переглянути. Це можна зробити шляхом перевірки компонента finalModel об’єкта modelFit. Щоб отримати доступ до цього компонента, потрібно просто використовувати знак долара, а потім finalModel. Цей компонент надає фактичні підігнані значення, отримані для моделі GLM.

```
predictions <- predict(modelFit, newData=testing)
predictions
#  [1] spam    spam    spam    spam    spam    spam    spam    spam    spam
# [10] spam    spam    spam    spam    spam    spam    nonspam spam    spam
# [19] spam    nonspam spam    nonspam spam    spam    nonspam spam    nonspam
# [28] ...
# ...
```

Щоб прогнозувати нові зразки, можна використати команду predict. Ця команда працює в уніфікованій структурі, вимагаючи лише modelFit, отриманого з функції train в caret, разом із даними, щодо яких потрібні прогнози. У цьому випадку нові дані відповідають даним тестування. Після виконання команда predict надає набір прогнозувань, що відповідають відповідям, полегшуючи оцінку підгонки моделі.

```
confusionMatrix(predictions, testing$type)
```

Один із способів оцінити модель – обчислити матрицю невідповідностей за допомогою функції confusionMatrix. Функція потребує прогнозів, отриманих на основі підгонки моделі, і фактичних результатів на вибірці тестування, таких як тип повідомлення. Потім створюється таблиця з детальним описом випадків, коли прогноз збігається з фактичним результатом (істинно позитивні та істинно негативні), а де ні (хибно позитивні та хибно негативні). Крім того, вона надає зведені статистичні дані, включаючи точність і 95% довірчий інтервал для точності, а також такі показники, як чутливість і специфічність, пропонуючи комплексну оцінку підгонки моделі.

2. Нарізка даних

Нарізку даних можна використовувати або для побудови навчальних і тестових наборів на початку створення функції прогнозування, або для проведення перехресної перевірки чи бутстрепінгу в навчальному наборі для оцінки моделей.

```
library(caret)
library(kernlab)
data(spam)
inTrain <- createDataPartition(y=spam$type, p=0.75, list=FALSE)
training <- spam[inTrain,]
testing <- spam[-inTrain,]
```

Знову ж таки, для цього завдання буде використано пакет caret. Пакет caret завантажено, а також набір даних kernlab для доступу до даних spam. Цей набір даних включає в себе прогнозування того, чи є електронні листи спамом чи ні, на основі різних змінних, таких як частота слів і використання великих літер. Спочатку функція createDataPartition використовується для поділу набору даних на навчальний і тестовий набори. Вказуючи змінну результату, функція розподіляє 75% даних для навчального набору та 25% для тестового набору. Згодом змінна inTrain генерується як функція індикатора, полегшуючи піднабір наборів для навчання та тестування. Навчальний набір витягується шляхом вибірки фрейму даних spam за допомогою змінної inTrain, тоді як тестовий набір містить решту зразків, які не входять до навчального набору.

```
set.seed(32323)
```

```
folds <- createFolds(y=spam$type, k=10, list=TRUE, returnTrain=TRUE)
sapply(folds, length)
# Fold01 Fold02 Fold03 Fold04 Fold05 Fold06 Fold07 Fold08 Fold09 Fold10
#   4140   4142   4141   4140   4141   4141   4142   4141   4141   4140
folds[[1]][1:10]
# [1]  1  2  4  5  6  7  8  9 10 11
```

Один із методів розділення даних на навчальні та тестові набори був обговорений. Інший підхід передбачає використання перехресної перевірки, коли навчальний набір ділиться на кілька менших підмножин, відомих як K-згортки. Це можна зробити за допомогою функції createFolds у пакеті caret. Для функції потрібно вказати змінну результату, наприклад spam$type, і потрібну кількість згорток, наприклад 10. Встановивши list=TRUE, функція повертає кожен набір індексів, що відповідає згортці, як список. Користувачі також можуть вибрати, чи включати навчальний набір у вихідні дані. Після виконання функція розбиває набір даних на десять згорток, кожна з яких містить однакову кількість зразків. Згодом користувачі можуть перевірити склад кожної згортки, щоб визначити, які зразки належать до навчальних і тестових наборів у K-згортках.

```
set.seed(32323)
folds <- createFolds(y=spam$type, k=10, list=TRUE, returnTrain=FALSE)
sapply(folds, length)
# Fold01 Fold02 Fold03 Fold04 Fold05 Fold06 Fold07 Fold08 Fold09 Fold10
#    461    459    460    461    460    460    459    460    460    461
folds[[1]][1:10]
# [1]  3 19 33 38 44 51 66 67 72 83
```

Крім того, функції можна вказати повертати лише тестовий або навчальний набір. Наприклад, якщо встановити returnTrain=FALSE, буде повернуто лише зразки тестового набору. Отже, кожна згортка міститиме меншу кількість зразків, оскільки більшість віднесено до навчального набору. Потім користувачі можуть досліджувати зразки, які належать до набору для тестування в кожній згортці, наприклад у згортці 1.

```
set.seed(32323)
folds <- createResample(y=spam$type, times=10, list=TRUE)
sapply(folds, length)
# Resample01 Resample02 Resample03 Resample04 Resample05 Resample06 Resample07
#       4601       4601       4601       4601       4601       4601       4601
# Resample08 Resample09 Resample10
#       4601       4601       4601
folds[[1]][1:10]
# [1] 1 1 2 2 3 3 4 5 5 7
```

Перевибірку можна здійснити за допомогою функції createResample. Цей метод пропонує альтернативу повній перехресній перевірці, такій як повторна вибірка або бутстрепінг. Користувачі вказують бажану кількість ітерацій повторної вибірки та бажаний вихідний формат (список, вектор або матриця). Ця функція надає інформацію

про те, які зразки належать до кожної згортки. При повторній вибірці можуть виникнути дублікати через заміну під час процесу відбору. Наприклад, у першій згортці зразок 3 може з'являтися кілька разів у результаті повторного відбору із заміною.

```
set.seed(32323)
tme <- 1:1000
folds <- createTimeSlices(y=tme, initialWindow=20, horizon=10)
names(folds)
# [1] "train" "test"
folds$train[[1]]
# [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
folds$test[[1]]
# [1] 21 22 23 24 25 26 27 28 29 30
```

Ви можете використовувати це для створення часових зрізів. Під час аналізу даних для цілей прогнозування користувачі можуть генерувати часові зрізи для вивчення безперервних значень у часі. Наприклад, шляхом створення вектора часу в діапазоні від 1 до 1000 з інтервалами в 1 і визначенням розміру вікна приблизно 20 зразків, а потім прогнозування наступних 10 зразків. У початковому навчальному наборі було б 20 значень, що становлять перше вікно, яке використовується для прогнозування. Згодом тестовий набір міститиме наступні 10 зразків, від 21 до 30, які представляють наступний часовий зріз. Цей процес продовжується, генеруючи послідовні набори чисел для використання часових коливань даних для прогнозного аналізу.

3. Параметри навчання

Зараз буде обговорено деякі параметри контролю навчання, доступні під час навчання з пакетом caret. Приклад спаму буде використано знову, щоб проілюструвати, як працюють ці ідеї. Як правило, під час підгонки моделі можна використовувати функцію train, встановлюючи для більшості параметрів значення за замовчуванням, вибрані функцією, за винятком, можливо, методу, що використовується для підгонки, і набору даних, який буде використано.

```
train(
  x,
  y,
  method = "rf",
  preProcess = NULL,
  ...,
  weights = NULL,
  metric = ifelse(is.factor(y), "Accuracy", "RMSE"),
  maximize = ifelse(metric %in% c("RMSE", "logLoss", "MAE", "logLoss"), FALSE,
TRUE),
  trControl = trainControl(),
  tuneGrid = NULL,
  tuneLength = ifelse(trControl$method == "none", 1, 3)
```

```
)
```

Можна глибше заглибитися в процес навчання, використовуючи різні варіанти. Наприклад, параметр preProcess пропонує кілька варіантів попередньої обробки. Крім того, вагові коефіцієнти можна встановити для конкретних спостережень із збільшенням або зменшенням ваги, що особливо корисно для вирішення незбалансованих навчальних наборів. Параметр metric визначає метрику за замовчуванням для факторних змінних (категоріальних) і безперервних змінних, причому точність і середньоквадратична помилка є значеннями за замовчуванням відповідно. Крім того, численні інші параметри управління можна налаштувати за допомогою параметра trControl, що потребує виклику функції trainControl.

Параметри метрики вбудовано у функцію train для безперервних результатів. Опції включають RMSE (середньоквадратична помилка) і R-квадрат. R-квадрат представляє лінійну відповідність між прогнозованими та фактичними змінними, що зазвичай використовується в лінійній регресії. Однак це може бути менш застосовним у нелінійних моделях, таких як випадкові ліси. Для категоріальних результатів мірою за замовчуванням є точність, яка вказує на частку правильних прогнозів. Крім того, ви можете вибрати коефіцієнт каппа, більш складну міру конкордансу, яка часто використовується в різних змаганнях.

```
trainControl(
  method = "boot",
  number = ifelse(grepl("cv", method), 10, 25),
  repeats = ifelse(grepl("[d_]cv$", method), 1, NA),
  p = 0.75,
  search = "grid",
  initialWindow = NULL,
  horizon = 1,
  fixedWindow = TRUE,
  skip = 0,
  verboseIter = FALSE,
  returnData = TRUE,
  returnResamp = "final",
  savePredictions = FALSE,
  classProbs = FALSE,
  summaryFunction = defaultSummary,
  selectionFunction = "best",
  preProcOptions = list(thresh = 0.95, ICAcomp = 3, k = 5, freqCut = 95/5,
uniqueCut = 10, cutoff = 0.9),
  sampling = NULL,
  index = NULL,
  indexOut = NULL,
  indexFinal = NULL,
  timingSamps = 0,
  predictionBounds = rep(FALSE, 2),
  seeds = NA,
```

```
  adaptive = list(min = 5, alpha = 0.05, method = "gls", complete = TRUE),
  trim = FALSE,
  allowParallel = TRUE
)
```

Аргумент trainControl дозволяє бути набагато точнішими щодо способу навчання моделей. Ви можете вказати метод повторної вибірки даних, будь то бутстрепінг чи перехресна перевірка. Крім того, ви можете визначати кількість ітерацій для бутстрепінгу або перехресної перевірки. За потреби ви також можете встановити кількість повторів для всього процесу для повторної перехресної перевірки. Можна вказати такі параметри, як розмір навчального набору, параметр p та інші параметри, що стосуються конкретної проблеми. Наприклад, під час аналізу даних про хід часу параметр initialWindow визначає розмір навчального набору даних, тоді як параметр horizon вказує кількість часових точок для прогнозування. Крім того, ви можете повертати фактичні прогнози з кожної ітерації під час створення моделі та налаштовувати функцію зведення. Також можна налаштувати параметри попередньої обробки, межі прогнозування та початкові параметри для шарів повторної вибірки. Встановлення початкових значень для шарів повторної вибірки особливо корисно для розпаралелювання обчислень на кількох ядрах, що може значно підвищити ефективність обчислень під час навчання моделей на великих наборах даних із численними предикторами. Хоча детальне висвітлення паралельної обробки виходить за рамки цього обговорення, варто відзначити її потенційну корисність у робочих процесах аналізу.

Для повторної вибірки доступні різноманітні методи, які передаються у функцію trainControl. Ви можете обрати стандартний бутстрепінг або бутстрепінг, який регулює повторну повторну вибірку кількох зразків, зменшуючи зміщення. Перехресна перевірка – ще один варіант, який обговорювався в попередній лекції. Повторна перехресна перевірка дозволяє здійснювати підперехресну перевірку з різними випадковими вибірками. Також доступна перехресна перевірка за винятком одного. Хоча вона тягне за собою компроміс між великою та малою кількістю згорток, можна вказати кількість зразків бутстрепінгу або підвибірок, а також кількість повторів для підвибірки. Хоча значень за замовчуванням зазвичай достатньо, у випадках із великою кількістю зразків або моделей, які вимагають значного налаштування параметрів, може знадобитися коригування таких параметрів, як кількість зразків перехресної перевірки або бутстрепінгу.

Нарешті, важливий аспект навчання цих моделей включає в себе встановлення початкового значення. Більшість цих процедур покладаються на випадкову повторну вибірку даних. Якщо код повторно запускається, результат може дещо відрізнятися через випадкову вибірку під час перехресної перевірки. Встановлення початкового значення гарантує, що кожного разу генеруватимуться однакові випадкові числа. Хоча цю концепцію може бути складно зрозуміти, ідея полягає в тому, що комп'ютер генерує псевдовипадкові числа, а встановлення початкового значення забезпечує генерацію тієї самої послідовності псевдовипадкових чисел. Встановлення початкового значення часто дуже корисно для отримання повторюваних результатів протягом усієї процедури. Якщо використовується паралельне обчислення, початкове значення для кожної повторної вибірки також можна встановити за допомогою початкового аргументу у функції trainControl. Встановлення початкового значення для кожної повторної вибірки є особливо корисним для паралельних підгонок, хоча це часто непотрібно для непаралельної обробки.

```
set.seed(1235)
modelFit2 <- train(type ~., data=training, method="glm")
modelFit2
# Generalized Linear Model
#
# 3451 samples
#   57 predictor
#    2 classes: 'nonspam', 'spam'
#
# No pre-processing
# Resampling: Bootstrapped (25 reps)
# Summary of sample sizes: 3451, 3451, 3451, 3451, 3451, 3451, ...
# Resampling results:
#
#   Accuracy   Kappa
#   0.9225284  0.8372078
set.seed(1235)
modelFit3 <- train(type ~., data=training, method="glm")
modelFit3
# Generalized Linear Model
#
# 3451 samples
#   57 predictor
#    2 classes: 'nonspam', 'spam'
#
# No pre-processing
# Resampling: Bootstrapped (25 reps)
# Summary of sample sizes: 3451, 3451, 3451, 3451, 3451, 3451, ...
# Resampling results:
#
#   Accuracy   Kappa
#   0.9225284  0.8372078
```

Ось приклад цього. Якщо ви встановите початкове значення за допомогою функції set.seed у R і призначите йому ціле число, це встановить початкове значення, яке забезпечує послідовність під час виконання аналізу. Отже, під час підгонки моделі, вибірки бутстрепінгу будуть згенеровані на основі випадкових чисел, отриманих із цього початкового значення. Наступне скидання вихідного коду та повторна підгонка моделі, цього разу з іншим номером, наприклад, modelFit3 замість modelFit2, дасть ті самі зразки бутстрепінгу та показники точності. Це особливо важливо під час навчання моделей і наміру поділитися навчальним набором даних з іншими, щоб забезпечити узгоджені результати під час запуску того самого коду.

4. Зображення предикторів на графіку

Одним із найважливіших компонентів побудови алгоритму машинного навчання або моделі прогнозування є розуміння того, як насправді виглядають дані та як вони взаємодіють один з одним. Таким чином, найкращим підходом є побудова графіка даних, зокрема предикторів. Для цього прикладу будуть використані дані Wage. Дані Wage отримані з пакета ISLR і представлені в книзі “Introduction to Statistical Learning”. Мета полягає в тому, щоб вивчити ці дані та дослідити їх корисність для прогнозування.

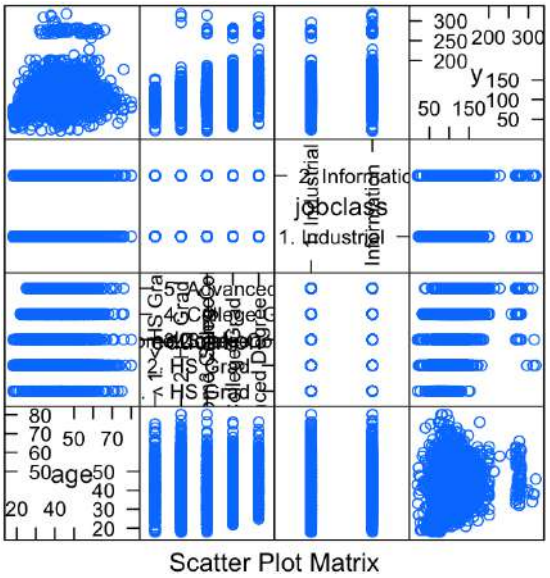
```
library(ISLR)
library(ggplot2)
library(caret)
data(Wage)
summary(Wage)
#       year           age                     maritl           race
#  Min.   :2003   Min.   :18.00   1. Never Married: 648   1. White:2480
#  1st Qu.:2004   1st Qu.:33.75   2. Married      :2074   2. Black: 293
#  Median :2006   Median :42.00   3. Widowed      :  19   3. Asian: 190
#  Mean   :2006   Mean   :42.41   4. Divorced     : 204   4. Other:  37
#  3rd Qu.:2008   3rd Qu.:51.00   5. Separated    :  55
#  Max.   :2009   Max.   :80.00
#
#               education                     region               jobclass
#  1. < HS Grad      :268   2. Middle Atlantic   :3000   1. Industrial :1544
#  2. HS Grad        :971   1. New England       :   0   2. Information:1456
#  3. Some College   :650   3. East North Central:   0
#  4. College Grad   :685   4. West North Central:   0
#  5. Advanced Degree:426   5. South Atlantic    :   0
#                           6. East South Central:   0
#                           (Other)              :   0
#             health      health_ins      logwage           wage
#  1. <=Good     : 858   1. Yes:2083   Min.   :3.000   Min.   : 20.09
#  2. >=Very Good:2142   2. No : 917   1st Qu.:4.447   1st Qu.: 85.38
#                                      Median :4.653   Median :104.92
#                                      Mean   :4.654   Mean   :111.70
#                                      3rd Qu.:4.857   3rd Qu.:128.68
#                                      Max.   :5.763   Max.   :318.34
```

Щоб завантажити пакет ISLR, його потрібно спочатку встановити, а потім завантажити. Крім того, пакет ggplot2 буде завантажено для побудови графіків і пакет caret для побудови моделі. Дані Wage доступні в пакеті ISLR і можуть бути завантажені за допомогою функції data. Підсумок даних Wage розкриває різні змінні, такі як рік збору даних, вік, сімейний стан, раса, освіта, регіон (зокрема Середньоатлантичний регіон), клас роботи та стан здоров'я. Цей підсумок містить початкове уявлення про набір даних, наприклад, включення лише осіб чоловічої статі та географічний регіон збору даних – Середньоатлантичний.

```
inTrain <- createDataPartition(y=Wage$wage, p=0.7, list=FALSE)
training <- Wage[ inTrain,]
testing <- Wage[-inTrain,]
dim(training)
# [1] 2102   11
dim(testing)
# [1] 898  11
```

Потім будуються навчальний і тестовий набори. Перед будь-яким дослідженням тестовий набір відокремлюється, і він залишається недоторканим до перевірки даних наприкінці процесу побудови моделі, де він використовується лише один раз.

```
featurePlot(x=training[,c("age", "education", "jobclass")], y=training$wage,
plot="pairs")
```

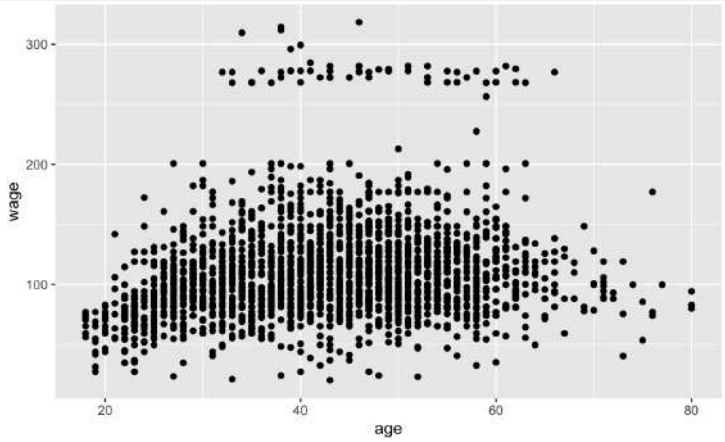


Scatter Plot Matrix

Вся побудова графіків ведеться в рамках навчального набору. Прикладом цього є використання featurePlot, функції, що надається пакетом caret. Цей графік відображає один проти одного всі властивості, які на перший погляд можуть здатися складними. При визначенні результату, позначеного як y, і дослідженні різних змінних, таких як вік, освіта та клас роботи, кожен графік демонструє зв'язок між результатом і

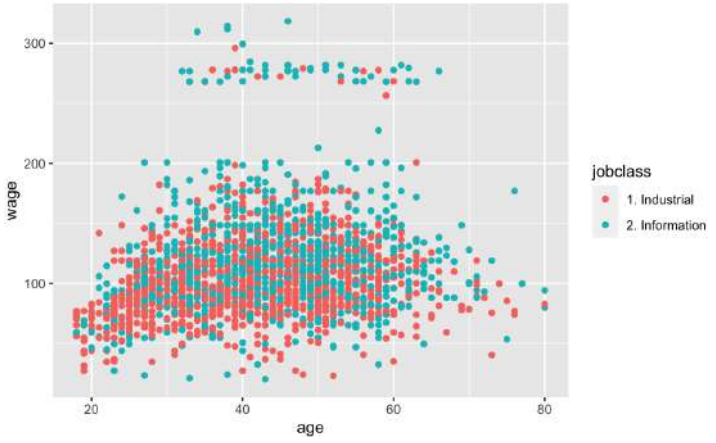
конкретною змінною. Наприклад, один графік ілюструє зв’язок між класом роботи та результатом, тоді як інший зображує клас роботи проти освіти. Спостереження за цими графіками дозволяє ідентифікувати змінні, які виявляють помітний зв’язок із результатом. Наприклад, можна спостерігати помітну тенденцію між освітою та заробітною платою. Цей метод пропонує всебічне уявлення про дані та допомагає визначити важливі змінні.

```
qplot(age, wage, data=training)
```



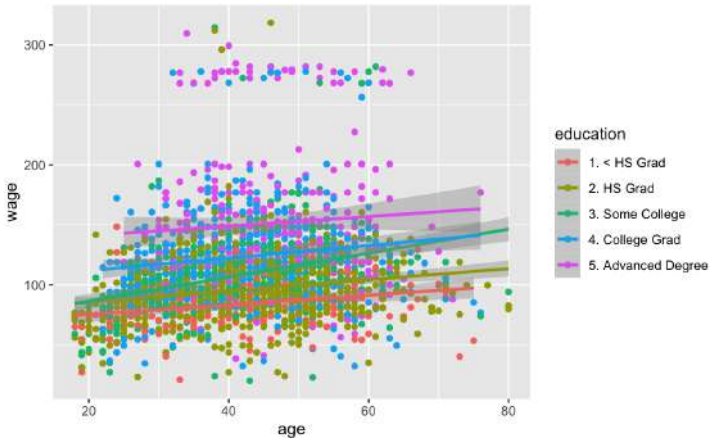
Іншим інструментом для використання є або функція qplot у пакеті ggplot2, або просто функція plot у базовому R. Наприклад, побудова графіка віку та заробітної плати показує потенційну тенденцію між двома змінними. Однак важливо звернути увагу на специфічні моделі, які можуть виникнути з таких графіків. У цьому випадку, здається, існує окремий кластер спостережень, який демонструє інший зв’язок між віком і заробітною платою порівняно з іншими кластерами. Розуміння причини цих аномалій має вирішальне значення перед тим, як продовжити створення моделі.

```
qplot(age, wage, colour=jobclass, data=training)
```



Один підхід може включати використання пакета ggplot2 для кольорового кодування графіка на основі різних змінних. Наприклад, при побудові вікової

залежності від заробітної плати з віком на осі x і заробітною платою на осі y графік можна розфарбувати за класом роботи. Цього можна досягти, вказавши параметр colour у функції ggplot. Графік показує, що більшість осіб у окремому кластері працюють на інформаційних роботах, а не на промислових. Це розуміння може допомогти пояснити невідповідність між цими двома помітними кластерами спостережень, тим самим допомагаючи ідентифікувати впливові змінні для моделі. Крім того, для подальшого аналізу можна включити згладжувачі регресії.




```
qq <- qplot(age, wage, colour=education, data=training)
qq + geom_smooth(method="lm", formula=y~x)
```

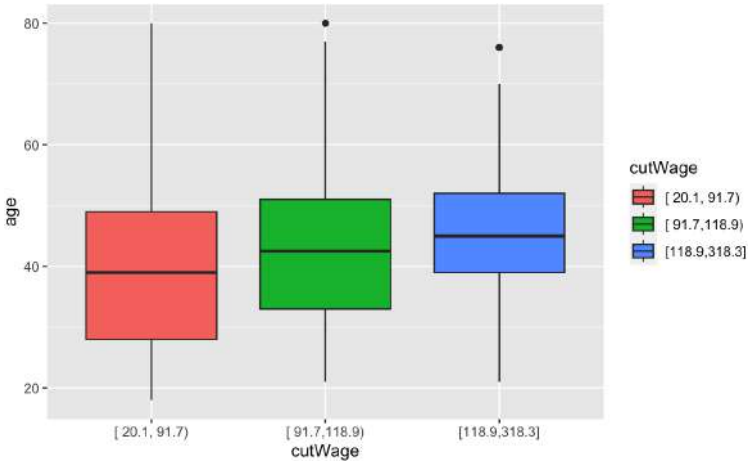
У цьому прикладі було створено графік залежності віку від заробітної плати з даними, розфарбованими за рівнем освіти. Потім функція geom_smooth використовується для застосування лінійного згладжування до даних. Ця функція відповідає лінійній моделі для кожного окремого класу освіти. Наприклад, фіолетова лінія позначає осіб із вченим ступенем, тоді як зелена лінія відповідає тим, хто має вищу освіту тощо. Цей аналіз допомагає визначити, чи існують різні співвідношення між віком і заробітною платою в різних освітніх групах.

```
library(Hmisc)
cutWage <- cut2(training$wage, g=3)
table(cutWage)
# cutWage
# [ 20.1, 91.7) [ 91.7,118.9) [118.9,318.3]
#           702           722           678
```

Іншим корисним підходом є категоризація змінних, таких як змінна wage, на різні групи, оскільки різні категорії можуть демонструвати різні взаємозв’язки. Це можна зробити за допомогою функції cut2 з пакету Hmisc. Вказуючи кількість груп за допомогою параметра g, функція ділить набір даних на фактори на основі груп квантилів. Наприклад, спостереження, що знаходяться в діапазоні від 20,1 до 91,7 за змінною заробітної плати, будуть віднесені до одного рівня факторів, тоді як значення

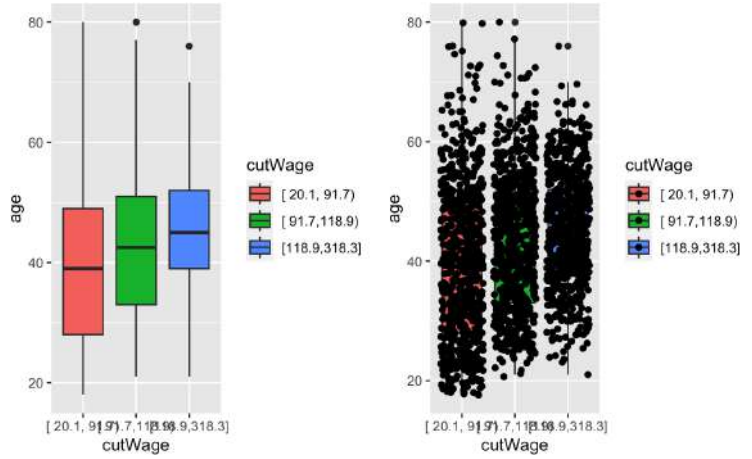
від 91,7 до 118,9 належатимуть до іншої групи, і так далі. Ця категоризація дозволяє створювати різні типи графіків для подальшого аналізу даних.

```
p1 <- qplot(cutWage, age, data=training, fill=cutWage, geom=c("boxplot"))
p1
```



Тепер, якщо ви бажаєте побудувати графік груп заробітної плати за віком, ви можете знову використати qplot, цього разу вказавши геометрію boxplot. Якщо вказати графік різних груп заробітної плати в залежності від віку, то можна виявити більш чіткі тенденції. Цей підхід може запропонувати більш чіткий погляд на зв'язок між віком і заробітною платою.

```
library(gridExtra)
p2 <- qplot(cutWage, age, data=training, fill=cutWage, geom=c("boxplot",
"jitter"))
grid.arrange(p1, p2, ncol=2)
```



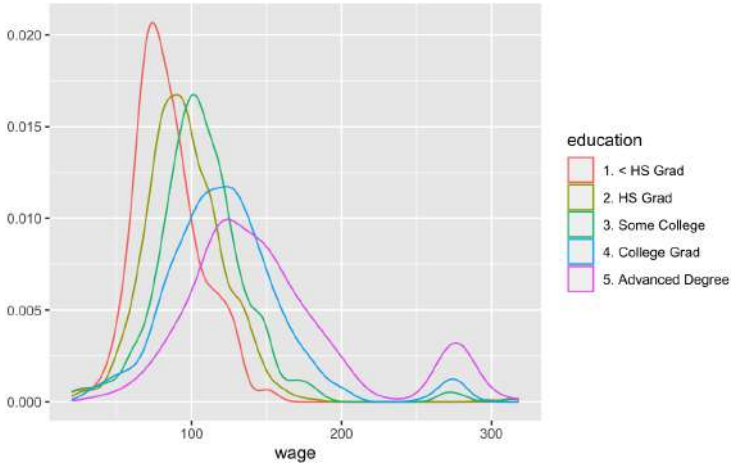
Інший варіант, який варто розглянути, це додавання самих точок поверх коробкових графіків. Це може бути корисним, оскільки коробкові графіки іноді не приховують кількості представлених точок. Щоб досягти цього, можна використовувати як коробковий графік, так і джиттер, вказавши їх на графіку. Використовуючи сітку, можна згенерувати два графіки поряд: один лише з коробковим

графіком (p1 із попереднього прикладу), а інший із накладеними точками (p2 із поточного прикладу). Спостереження за численними точками в кожній коробці свідчить про потенційно справжню тенденцію. І навпаки, якщо видно лише кілька точок, це вказує на те, що конкретне вікно може бути неповним представленням даних.

```
t1 <- table(cutWage, training$jobclass)
t1
# cutWage         1. Industrial 2. Information
#   [ 20.1, 91.7)           459            243
#   [ 91.7,118.9)           387            335
#   [118.9,318.3]           276            402
prop.table(t1, 1)
# cutWage         1. Industrial 2. Information
#   [ 20.1, 91.7)     0.6538462      0.3461538
#   [ 91.7,118.9)     0.5360111      0.4639889
#   [118.9,318.3]     0.4070796      0.5929204
```

Ще один корисний прийом полягає у використанні змінної cut, яка представляє факторизовану версію безперервної змінної, для створення таблиць даних. Наприклад, можна створити таблицю порівняння цієї факторної версії заробітної плати з класом роботи. Вивчаючи таблицю, можна спостерігати такі закономірності, як більше робочих місць у промисловості з нижчою категорією заробітної плати порівняно з робочими місцями в галузі інформації. І навпаки, для вищих класів заробітної плати є менше робочих місць у промисловості та більше робочих місць у сфері інформації. Крім того, функція prop.table може бути використана для отримання пропорцій у кожній групі. Якщо вказати 1, вона обчислить пропорцію в кожному рядку, а вказавши 2, обчислить пропорцію в кожному стовпці. Наприклад, це показує, що 65% низькооплачуваних робіт відповідають посадам у промисловості, тоді як 35% відповідають роботам у сфері інформації. Цей підхід дає змогу зрозуміти, як ці пропорції змінюються залежно від рівня заробітної плати.

```
qplot(wage, colour=education, data=training, geom="density")
```



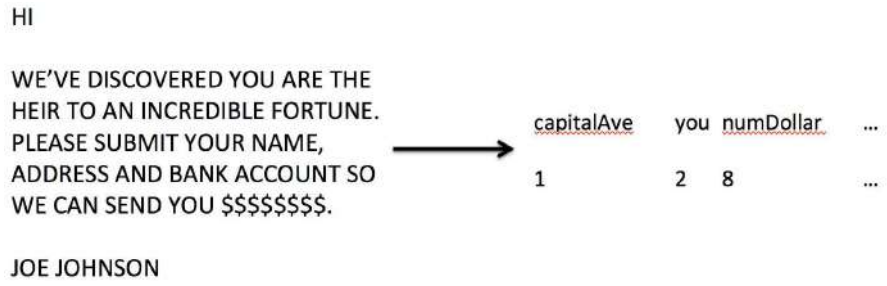
Нарешті, ви можете використовувати графіки густини для візуалізації значень безперервних предикторів. Використовуючи функцію qplot, змінна заробітної плати будується проти освіти на графіку густини. Це представлення показує, де знаходиться більшість даних. Вісь x представляє заробітну плату, тоді як вісь y позначає частку змінної, що потрапляє в кожне поле осі x. Наприклад, випускники середньої школи, як правило, мають більше значень у нижньому діапазоні, тоді як ті, хто має вчений ступінь, демонструють вищі значення. Крім того, існує окрема група, виділена синім кольором, до якої входять як власники вчених ступенів, так і випускники коледжів, які, як правило, мають дуже високі заробітні плати. Діаграми густини пропонують інформацію, яку коробкові графіки можуть не охопити, і дозволяють легше накладати кілька розподілів, що робить їх особливо корисними для аналізу розподілів різних груп.

Важливо створювати графіки виключно на основі навчальних даних. Тестовий набір не слід використовувати для дослідження, щоб запобігти перепідгонці, як обговорювалося раніше. Вивчаючи ці графіки, люди повинні стежити за дисбалансами в прогнозних результатах. Якщо всі предиктори переважно показують одну групу результатів над іншою, це свідчить про сильний предиктор. І навпаки, якщо один результат має лише кілька випадків порівняно з іншим, стає складно створити точний класифікатор між двома класами. Крім того, викиди або незвичайні кластери даних можуть вказувати на відсутність змінних. Крім того, слід звернути увагу на групи точок, які не враховуються жодними предикторами, а також викривлені змінні. Викривлені змінні можуть вимагати трансформації для досягнення більш нормального розподілу, особливо для регресійних моделей, хоча це може бути не настільки критично для методів машинного навчання.

5. Створення коваріантів

Коваріанти, які також називають предикторами або властивостями, є змінними, включеними в модель для прогнозування бажаного результату. Існує два етапи створення коваріантів або розробки властивостей. Спочатку необроблені дані, такі як зображення, текстові файли або веб-сайти, повинні бути перетворені в придатні для використання предиктори. Цей процес включає узагальнення інформації в кількісні або якісні змінні, щоб полегшити підгонку стандартних алгоритмів машинного навчання. Розглянемо, наприклад, електронний лист. Непрактично безпосередньо вводити електронний лист у функцію прогнозування через його природу вільного тексту. Натомість генеруються характеристики для опису електронного листа, як-от частота

великих літер, повторення певних слів, як-от “you”, або кількість знаків долара, які можуть вказувати на спам. Цей крок вимагає ретельного розгляду структури даних для ефективного вилучення відповідної інформації з мінімальними змінними.




```
library(kernlab)
data(spam)
spam$capitalAveSq <- spam$capitalAve^2
```

Наступний етап передбачає перетворення чистих коваріантів у більш корисні змінні. Наприклад, середня кількість великих літер в електронному листі може прямо не співвідноситися з бажаним результатом. Тому такі перетворення, як зведення в квадрат або куб цих змінних, виконуються для підвищення їхньої прогностичної корисності. Використовуючи дані kernlab і набір даних spam як приклад, середню частку капіталу можна звести в квадрат, щоб створити нову змінну capitalAvSq, яка може виявитись корисною в наступних алгоритмах прогнозування. Ці кроки є невід’ємною частиною процесу створення коваріантів.

На першому кроці коваріант необроблених даних значною мірою залежить від застосування. Наприклад, у контексті електронних листів це може включати вилучення частоти повторень слів. У випадку з голосовими даними це може містити такі характеристики, як частота або тембр. Для зображень фокус може бути на ідентифікації таких компонентів обличчя, як ніс, вуха чи очі. Вибір властивостей значною мірою залежить від конкретного застосування та вимагає тонкого балансу між узагальненням і мінімізацією втрати інформації. Ефективні властивості фіксують релевантну інформацію, відкидаючи зайві деталі. Отже, уважний розгляд є важливим для вибору властивостей, які прояснюють більшість тонкощів необроблених даних.

Деякі ілюстративні приклади включають визначення частоти слів або фраз у текстових файлах, виявлення країв, кутів, плям або виступів на зображеннях, оцінку кількості та типів зображень, розташування кнопок, кольорів і відео на веб-сайтах. Цей процес має вирішальне значення в різних областях, як-от веб-розробка, де A/B-тестування відіграє ключову роль в оптимізації дизайну веб-сайту для максимального залучення користувачів або продажів. Особливості, пов’язані з

окремими особами, можуть включати такі атрибути, як зріст, вага чи колір волосся. По суті, будь-яке стиснуте представлення необроблених даних може слугувати потенційною властивістю. Однак визначення відповідних коваріантів часто вимагає поєднання наукового досвіду та бізнес-розуміння, адаптованого до конкретної проблеми.

Хоча автоматизація може допомогти у вилученні властивостей, потрібна обережність, оскільки автоматизовані методи не завжди дають оптимальні результати. Деякі властивості можуть добре працювати в навчальному наборі, але не можуть ефективно узагальнюватися в нових наборах даних, що призводить до низької продуктивності моделі. Тому доцільно зробити помилку, спочатку створивши більше властивостей і згодом удосконаливши їх у процесі побудови моделі. Цей підхід забезпечує більшу гнучкість і гарантує, що відповідна інформація зберігається, а нерелевантні властивості відфільтровуються.

Другий рівень передбачає роботу з очищеними коваріантами, які є властивостями, уже отриманими з набору даних, для створення нових коваріантів. Як правило, це передбачає застосування перетворень або функцій до існуючих коваріантів, що може виявитися корисним під час побудови моделі прогнозування. Такі перетворення часто більш критичні для таких методів, як регресія або метод опорних векторів, які більшою мірою покладаються на розподіл даних. Навпаки, вони можуть бути менш важливими для таких методів, як дерева класифікації, які віддають пріоритет класифікації на основі даних без суворих припущень моделі.

Незалежно від використовуваного методу, як правило, рекомендується забезпечити включення відповідних коваріантів. При створенні цих властивостей або перетворень необхідно виконувати цю задачу виключно в рамках навчального набору, дотримуючись фундаментального принципу машинного навчання. Розробка властивостей має відбуватися виключно в навчальному наборі, щоб запобігти перепідгонці. Згодом, при застосуванні функції прогнозування до тестового набору, застосовуються ті самі перетворення, щоб забезпечити узгодженість. У цьому відношенні цінним підходом є дослідницький аналіз, який передбачає створення графіків і таблиць для виявлення закономірностей варіацій у наборі даних і їхнього зв’язку з результатом.

У R, коли для такого аналізу використовуються такі пакети, як caret, нові коваріанти повинні бути включені в фрейми даних для використання в наступних

прогнозах. Крім того, надзвичайно важливо присвоїти розпізнавані імена цим новим змінним, щоб полегшити їх використання в наборі даних тестування.

```
library(ISLR)
library(caret)
data(Wage)
inTrain <- createDataPartition(y=Wage$wage, p=0.7, list=FALSE)
training <- Wage[inTrain,]
testing <- Wage[-inTrain,]
```

Ось приклад набору даних. Щоб гарантувати, що створення всіх коваріантів виконується в навчальному наборі, набір даних розділено на навчальний і тестовий набори за допомогою функції createDataPartition для створення індексів для обох наборів даних. Згодом дані розподіляються відповідно на навчальний та тестовий набори.

```
table(training$jobclass)
#  1. Industrial 2. Information
#           1109            993
dummies <- dummyVars(wage ~ jobclass, data=training)
head(predict(dummies, newdata=training))
#        jobclass.1. Industrial jobclass.2. Information
# 231655                      1                       0
# 161300                      1                       0
# 155159                      0                       1
# 11443                       0                       1
# 376662                      0                       1
# 302778                      0                       1
```

Одним із поширених підходів у машинному навчанні є перетворення якісних або факторних змінних у фіктивні змінні. Наприклад, розглянемо змінну jobclass в навчальному наборі, яка складається з двох рівнів. Спроба безпосередньо інтегрувати цю змінну в модель прогнозування є проблемою, оскільки її значення є категоріальним. Алгоритми прогнозування часто мають проблеми зі змінними якісної інформації під час прогнозування. Щоб вирішити цю проблему, можна використати функцію dummyVars у пакеті caret. Вказуючи модель (із wage як результатом і jobclass як предиктором), разом із навчальним набором, ця функція генерує дві нові змінні: індикатор для промислових робочих місць і індикатор для інформаційних робочих місць. У отриманих фіктивних змінних значення 1 означає наявність відповідного типу роботи, тоді як 0 вказує на її відсутність. Наприклад, якщо людина має промислову роботу, то показник для промисловості буде 1, а для інформації – 0, і навпаки. Цей процес перетворення перетворює якісні змінні на кількісні, полегшуючи їх використання в моделях прогнозування.

```
nsv <- nearZeroVar(training, saveMetrics=TRUE)
```

```
nsv
#            freqRatio percentUnique zeroVar   nzv
# year        1.080838    0.33301618   FALSE FALSE
# age         1.164384    2.90199810   FALSE FALSE
# maritl      3.184211    0.23786870   FALSE FALSE
# race        8.326923    0.19029496   FALSE FALSE
# education   1.427083    0.23786870   FALSE FALSE
# region      0.000000    0.04757374    TRUE  TRUE
# jobclass    1.116818    0.09514748   FALSE FALSE
# health      2.526846    0.09514748   FALSE FALSE
# health_ins  2.223926    0.09514748   FALSE FALSE
# logwage     1.024096   18.69647954   FALSE FALSE
# wage        1.024096   18.69647954   FALSE FALSE
```

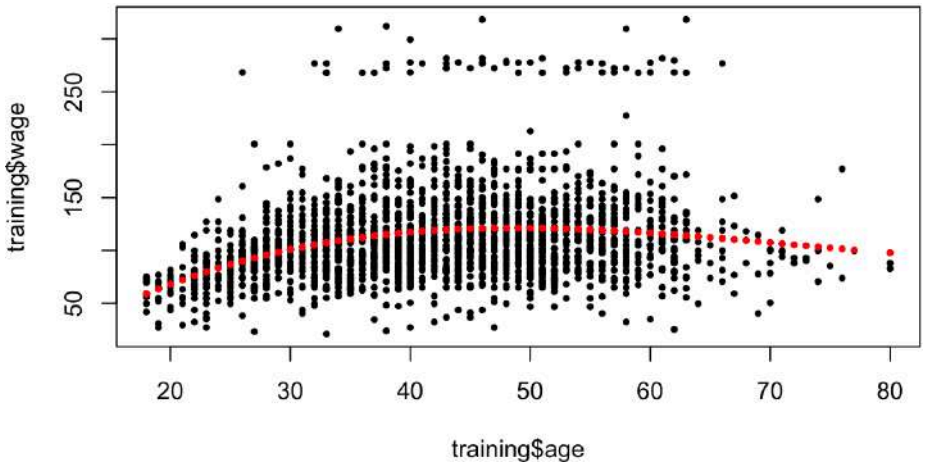
Інший випадок включає змінні з мінімальною мінливістю. Наприклад, якщо створено властивість для визначення того, чи електронний лист містить літери, майже всі електронні листи відповідатимуть цьому критерію, перетворюючи змінну на постійне істинне значення, позбавляючись мінливості. Отже, така змінна навряд чи слугуватиме корисним коваріантом. Щоб ідентифікувати ці змінні, можна використати функцію nearZeroVar у пакеті caret. Застосовуючи цю функцію до навчального набору даних, можна оцінити унікальність значень для кожної змінної. Наприклад, якщо змінна демонструє дуже низький відсоток унікальних значень, наприклад, змінна sex переважно містить чоловічу стать, це вказує на майже нульову дисперсію. Подібні змінні, включно з region, не мають суттєвої мінливості та непридатні для алгоритмів прогнозування. Таким чином, використання цього методу дозволяє швидко видалити менш значущі предиктори.

```
library(splines)
bsBasis <- bs(training$age, df=3)
bsBasis
#     1            2            3
#    [1,] 0.000000000 0.0000000000 0.000000e+00
#    [2,] 0.416337988 0.3211750193 8.258786e-02
#    [3,] 0.430813836 0.2910904300 6.556091e-02
#    [4,] 0.362525595 0.3866939680 1.374912e-01
#    ...
```

Інший підхід полягає у використанні базисних функцій замість підгонки прямих ліній при застосуванні лінійної регресії або узагальненої лінійної регресії для алгоритмів прогнозування. Цей метод дозволяє підігнати криві лінії, що може бути корисним у певних сценаріях. Базисні функції можна знайти в пакетах, таких як splines. Використовуючи функцію bs, ви можете створювати поліноміальні змінні. Наприклад, передаючи одну змінну і вказуючи поліном третього ступеня для такої змінної, як age, функція генерує матрицю з трьох стовпців. Перший стовпець представляє вихідні значення віку, тоді як другий і третій стовпці відповідають віку в квадраті та віку в кубі

відповідно. Включення цих коваріантів у модель дає змогу підібрати криві залежності між віком і результатом, на відміну від того, щоб покладатися виключно на змінну age в моделюванні лінійної регресії.

```
lml <- lm(wage ~ bsBasis, data=training)
plot(training$age, training$wage, pch=19, cex=0.5)
points(training$age, predict(lml, newdata=training), col="red", pch=19, cex=0.5)
```



Щоб проілюструвати цю концепцію, розглянемо наступний приклад. Лінійна модель підігнана, де wage є змінною результату. Використовуючи bsBasis, який охоплює всі предиктори, згенеровані поліноміальною моделлю (такі як вік, вік у квадраті та вік у кубі), будується модель. Згодом створюється графік, що зображує зв'язок між age і wage, з age на осі x і wage на осі y. Стає очевидним, що між цими двома змінними існує криволінійна залежність. Побудова графіка age проти прогнозованих значень, отриманих з лінійної моделі, яка включає поліноміальні члени, призводить до підгонки кривої до набору даних, а не прямої лінії. Цей підхід пропонує засоби для введення нових змінних шляхом включення більшої гнучкості в моделювання конкретних змінних.

```
predict(bsBasis, age=testing$age)
#                   1            2            3
#    [1,] 0.000000000 0.0000000000 0.000000e+00
#    [2,] 0.416337988 0.3211750193 8.258786e-02
#    [3,] 0.430813836 0.2910904300 6.556091e-02
#    [4,] 0.362525595 0.3866939680 1.374912e-01
#    ...
```

На тестовому наборі ви повинні прогнозувати ті самі змінні. Ця концепція має величезне значення в машинному навчанні під час створення нових коваріантів. Процедура передбачає створення коваріантів у тестовому наборі даних за допомогою того самого методу, який використовується для навчального набору. Це можна досягти шляхом прогнозування нового набору значень зі змінної, згенерованої за допомогою

функції bsBasis, зокрема для значень age в наборі для тестування. Ці значення згодом використовуються в моделі прогнозування під час тестування. І навпаки, створення нового набору предикторів шляхом безпосереднього застосування функції bsBasis до змінної age в тестовому наборі призведе до генерування непов'язаних змінних, які можуть внести зміщення.

Перший рівень створення властивостей в основному покладається на предметні знання або наукове розуміння. Щоб дослідити подальші шляхи для конкретних застосувань, які тут не розглядаються, або для новачків, корисним підходом є пошук методів вилучення ознак, адаптованих до типу даних, що аналізуються. Наприклад, пошук "вилучення властивостей для зображень" або "вилучення властивостей для голосу" може дати цінну інформацію. Бажано зосередитися на визначенні відмінних характеристик, які можуть відрізнятися в окремих зразках. Загалом спочатку краще створити більше властивостей, ніж потрібно, оскільки їх можна вдосконалити пізніше в процесі машинного навчання. У певних областях, як-от зображення та голоси, створення властивостей, які не є інтуїтивно зрозумілими, може бути важливим. Методи глибокого навчання пропонують засоби для автоматичного створення таких властивостей, і існують доступні ресурси для глибшого вивчення цього підходу. Однак дуже важливо підтримувати критичний настрій під час використання методів автоматичного створення властивостей, щоб забезпечити релевантність і узгодженість створених властивостей.

Другий рівень створення властивостей, що перетворює коваріанти на нові, можна здійснити за допомогою функціональних можливостей preProcess пакета caret. По суті, можна генерувати нові коваріанти, використовуючи різні функції, доступні в R, за умови, що вони логічні та розумні. Акцент залишається на проведенні широкого дослідницького аналізу та створенні численних графіків для визначення зв'язків між предикторами та результатами. Введення нових коваріантів життєздатне, якщо вважається, що вони покращують підгонку моделі. Однак слід бути обережним, щоб уникнути створення безглуздих властивостей. Перепідгонка викликає занепокоєння; якщо численні властивості пристосовані спеціально для навчального набору, вони можуть погано узагальнюватися на тестовому наборі. Тому доцільно відфільтрувати зайві властивості перед застосуванням алгоритму машинного навчання. Корисним ресурсом для базової попередньої обробки є пакет caret, який пропонує вичерпний посібник із методів попередньої обробки. Крім того, для побудови гнучких моделей

spline, як обговорювалося раніше, пакет caret забезпечує метод gam, що дозволяє згладжувати кілька змінних за допомогою окремих кривих для кожної змінної.

6. Прогнозування з регресією

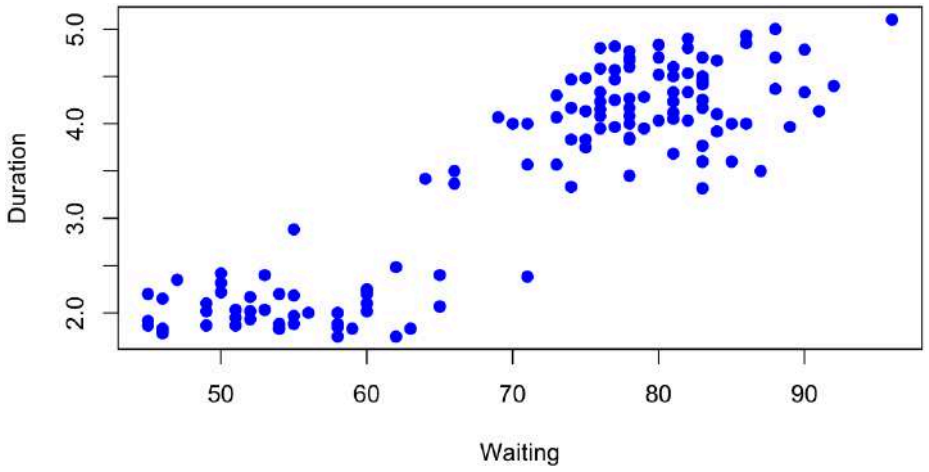
Ключова ідея тут полягає в тому, щоб підібрати просту модель регресії. По суті, концепція включає підгонку лінії до набору даних. Ця лінія визначається множенням набору коефіцієнтів на кожен із різних предикторів. Згодом отримують нові предиктори або коваріанти, які множать на коефіцієнти, оцінені за допомогою моделі прогнозування, що призводить до нового прогнозу для нового значення. Цей підхід корисний, коли лінійна модель є майже точною, тобто зв'язок між змінними можна моделювати лінійно. Його легко реалізувати та відносно легко інтерпретувати порівняно з багатьма іншими алгоритмами машинного навчання, оскільки він передбачає підгонку ліній до набору даних, які відносно легко інтерпретувати. Однак він може працювати погано в нелінійних налаштуваннях, тому його часто використовують разом з іншими алгоритмами машинного навчання в складних сценаріях.

```
library(caret)
data(faithful)
set.seed(333)
inTrain <- createDataPartition(y=faithful$waiting, p=0.5, list=FALSE)
trainFaith <- faithful[inTrain,]
testFaith <- faithful[-inTrain,]
head(trainFaith)
#    eruptions waiting
# 3      3.333      74
# 6      2.883      55
# 7      4.700      88
# 8      3.600      85
# 9      1.950      51
# 11     1.833      54
```

Аналіз буде зосереджений на даних про виверження гейзерів. Гейзери зазвичай мають період очікування між виверженнями, а також тривалість кожного виверження. Доступний набір даних містить інформацію про виверження конкретного гейзера Old Faithful, розташованого в Сполучених Штатах. Завантаживши пакет caret, можна легко отримати доступ до цього набору даних. Встановлення початкового значення гарантує, що всі наступні аналізи можна відтворити послідовно. Після цього створюється навчальний і тестовий набір відповідно до стандартної практики побудови моделі. Навчальний набір використовується для побудови моделей, тоді як тестовий набір використовується для застосування моделі. Перевіривши набір даних, стає очевидним,

що він містить лише дві змінні: час виверження та час очікування. Ця простота робить його ідеальним прикладом для аналізу.

```
plot(trainFaith$waiting, trainFaith$eruptions, pch=19, col="blue",
xlab="Waiting", ylab="Duration")
```



Під час побудови цих двох змінних час очікування відображається на осі x, а тривалість – на осі y. Очевидна лінійна залежність. Ви можете візуалізувати малювання лінії через точки даних, яка досить добре прогнозує час тривалості на основі часу очікування.

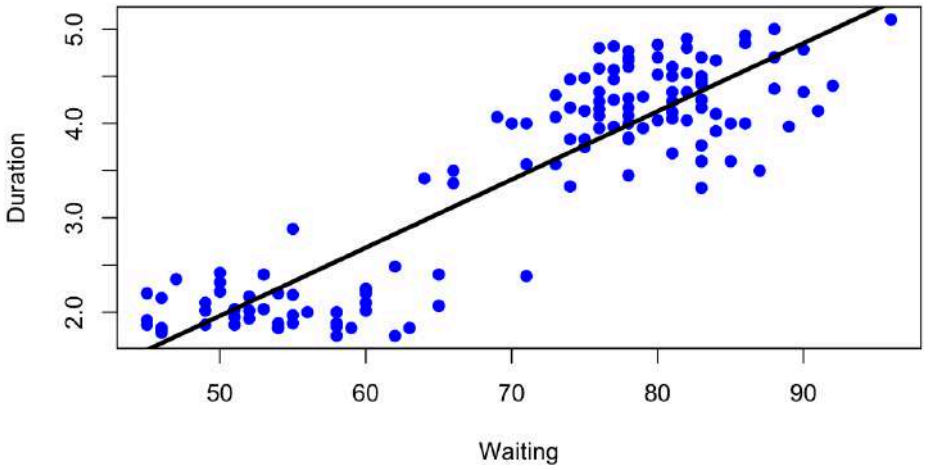
Щоб досягти цього, можна підібрати формулу, що представляє лінію. Формула для лінії складається з члена перетину (константа) і коефіцієнта, помноженого на змінну-прогноз (у цьому випадку час очікування), а також член помилки. Хоча лінія, підігнана через дані, може досить добре апроксимувати співвідношення, важливо визнати, що точки можуть не ідеально вирівнюватися на лінії. Ця розбіжність пояснюється членом помилки в моделі, який представляє невиміряну або непояснену мінливість у зв’язку.

```
lm1 <- lm(eruptions ~ waiting, data=trainFaith)
summary(lm1)
# Call:
# lm(formula = eruptions ~ waiting, data = trainFaith)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -1.13375 -0.36778  0.06064  0.36578  0.96057
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) -1.648629   0.226603  -7.275 2.55e-11 ***
# waiting      0.072211   0.003136  23.026  < 2e-16 ***
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#
# Residual standard error: 0.4941 on 135 degrees of freedom
```

```
# Multiple R-squared:  0.7971,   Adjusted R-squared:  0.7956
# F-statistic: 530.2 on 1 and 135 DF,  p-value: < 2.2e-16
```

Команда lm в R використовується для побудови лінійної моделі. За допомогою lm змінна результату (тривалість виверження) прогнозується як функція змінної предиктора (час очікування). Після застосування цієї команду до навчальних даних, ви отримуєте зведення, зосереджуючись на оцінках коефіцієнтів. У цьому контексті оцінка перетину (β_0) і оцінка часу очікування (β_1) є ключовими. Щоб зробити новий прогноз, потрібно просто додати нове значення часу очікування у формулу: $-1,64 + 0.072 * waiting\ time$.

```
plot(trainFaith$waiting, trainFaith$eruptions, pch=19, col="blue",
xlab="Waiting", ylab="Duration")
lines (trainFaith$waiting, lm1$fitted, lwd=3)
```



Так виглядає підгонка моделі. Час очікування в залежності від тривалості виверження нанесено на графік разом із підігнаними значеннями, отриманими з об'єкта лінійної моделі, lm1$fitted. Підігнана функція надає підігнані значення, які потім зображаються на графіку проти змінної предиктора, що використовується в моделі. У цьому випадку час очікування залежить від тривалості. Точки на графіку представляють фактичні точки даних, а чорна лінія – підігнані значення. Як зазначалося раніше, лінія слугує досить хорошим представленням зв'язку між цими двома змінними, хоча вона не ідеальна і не точно перетинає всі точки даних.

```
coef(lm1)[1] + coef(lm1)[2] * 80
# (Intercept)
#    4.128276
```

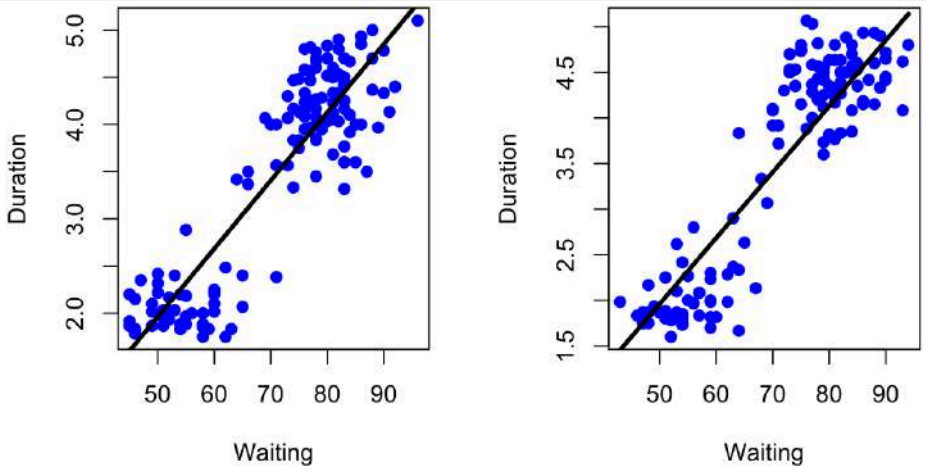
Щоб прогнозувати нову змінну, потрібно просто використовувати оцінені значення для β_0 і β_1, які зазвичай позначаються капелюхами над значеннями. Ці значення перемножуються за наведеною вище формулою. Важливо зазначити, що ця

формула не містить похибок, оскільки похибка для певного значення невідома, тому використовуються лише оцінювані компоненти. Щоб отримати ці значення з об’єкта лінійної моделі, ви можете використати команду coef, яка надає коефіцієнти, що представляють оцінки для цих двох змінних. Наприклад, coef(lm1)[1] забезпечить перетин, а coef(lm2)[2] запропонує оцінку нахилу, що стосується часу очікування. Припустимо, що є новий час очікування 80. Тоді кінцевий прогноз тривалості виверження дорівнює 4,128.

```
newdata <- data.frame(waiting=80)
predict(lm1, newdata)
#        1
# 4.128276
```

Іншим доступним варіантом є використання об’єкта lm для прогнозування. Немає необхідності кожного разу витягувати коефіцієнти та виконувати множення вручну. Створивши новий набір даних, зокрема фрейм даних із потрібним значенням для прогнозування (у цьому випадку час очікування 80), можна застосувати функцію predict. Передаючи їй підігнану модель із навчального набору та щойно створеного фрейму даних, функція забезпечує прогноз для нового значення. Варто зазначити, що ця команда predict використовує ту саму формулу, що й у разі обчислення прогнозу вручну.

```
par(mfrow=c(1,2))
plot(trainFaith$waiting, trainFaith$eruptions, pch=19, col="blue",
xlab="Waiting", ylab="Duration")
lines(trainFaith$waiting, predict(lm1), lwd=3)
plot(testFaith$waiting, testFaith$eruptions, pch=19, col="blue", xlab="Waiting",
ylab="Duration")
lines(testFaith$waiting, predict(lm1, newdata=testFaith), lwd=3)
```



Ще один аспект, який слід враховувати, полягає в тому, що модель була побудована виключно на основі навчального набору, дотримуючись стандартної

практики. Продуктивність моделі на тестовому наборі вимагає оцінки. Було створено два різних набори, навчальний і тестовий, а також два графіки для порівняння. Лівий графік ілюструє зв'язок між часом очікування та тривалістю з використанням даних навчання, включаючи підгонку моделі. Як і очікувалося, модель досить добре узгоджується з навчальними даними, враховуючи, що вона була розроблена з використанням цього набору даних. Праворуч на графіку зображено тестовий набір даних. Тут прогнози випливають із моделі, побудованої на навчальному наборі. Примітно, що підгонка не така точна, як це спостерігається з даними навчання; помітні незначні відхилення. Однак ця невідповідність є очікуваною, оскільки тестовий набір містить різні дані. Тим не менш, модель все ще фіксує загальну тенденцію або варіацію, пов'язану з часом очікування.

```
# Calculate RMSE on training
sqrt(sum((lm1$fitted - trainFaith$eruptions)^2))
# [1] 5.740844
```

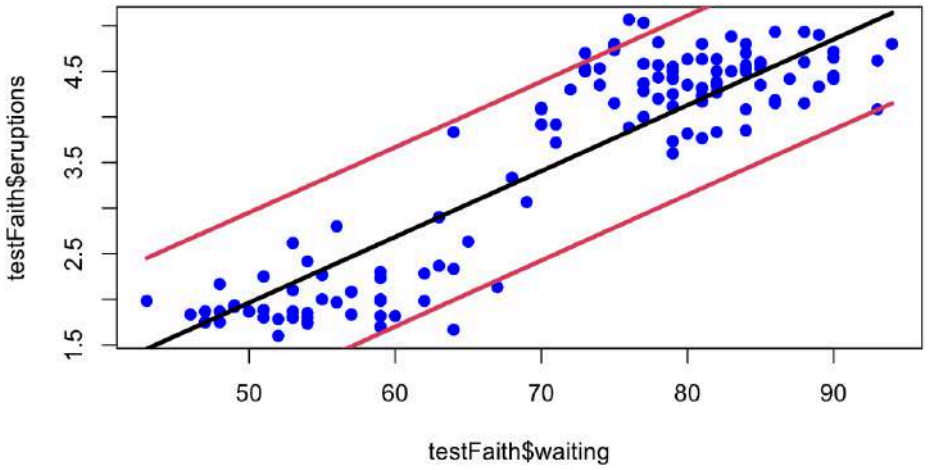
Наступним кроком є обчислення помилок навчального та тестового набору. Щоб отримати помилку навчального набору, витягують підігнані значення, позначені як lm1$fitted, де lm1 представляє підігнаний об'єкт моделі. Ці підігнані значення відповідають прогнозам, створеним на навчальному наборі. Далі фактичні значення тривалості виверження в навчальному наборі віднімаються з прогнозованих значень, зводяться в квадрат, підсумовуються, а потім береться квадратний корінь, що призводить до середньоквадратичної помилки (RMSE). Ця метрика по суті кількісно визначає близькість підігнаних значень до фактичних значень. Для цього показника помилки отримано значення 5,853.

```
# Calculate RMSE on test
sqrt(sum((predict(lm1, newdata=testFaith) - testFaith$eruptions)^2))
# [1] 5.853745
```

Подібним чином можна обчислити середню квадратичну помилку для тестового набору. Щоб досягти цього, прогнози робляться з використанням об'єкта lm, встановленого на навчальному наборі, але цього разу з новим набором даних – тестовим набором даних. Згодом фактичні значення віднімаються з прогнозованих значень, зводяться в квадрат, підсумовуються, а потім береться квадратний корінь. Оскільки тестовий набір не використовувався під час процесу побудови моделі, ця оцінка помилки забезпечує більш реалістичну оцінку RMSE, яка спостерігалася б на новому наборі даних, на відміну від значення, отриманого на навчальному наборі. Як зазвичай, помилка тестового набору зазвичай більша, ніж помилка навчального набору,

що відображає додаткову помилку та мінливість, які виникають під час переходу до нового набору даних, відому як помилка поза вибіркою.

```
pred1 <- predict(lm1, newdata=testFaith, interval="prediction")
ord <- order (testFaith$waiting)
plot(testFaith$waiting, testFaith$eruptions, pch=19, col="blue")
matlines(testFaith$waiting[ord], pred1[ord,], type="l", , col=c(1,2,2),
lty=c(1,1,1), lwd=3)
```



Ще однією перевагою використання лінійного моделювання для прогнозування є можливість обчислювати інтервали прогнозування. Використовуючи лінійну модель, побудовану на навчальному наборі даних, можна створити новий набір прогнозів для тестового набору даних. Вказуючи включення інтервалу прогнозування через аргумент, переданий функції прогнозування, можна отримати цю додаткову інформацію. Побудова графіка часу очікування даних для тестування проти часу виверження та накладання прогнозованих значень, представлених чорною лінією, разом із інтервалом прогнозування, позначеним двома червоними лініями, дає уявлення про діапазон очікуваних значень. Інтервал прогнозування означає область, де очікується потрапляння більшості прогнозованих значень, якщо лінійна модель вірна. Це не лише забезпечує єдиний прогноз, але й ілюструє потенційний діапазон прогнозів, пропонуючи цінну інформацію про прогнозну ефективність моделі на нових даних.

```
modFit <- train(eruptions ~ waiting, data=trainFaith, method="lm")
summary(modFit$finalModel)
# Call:
# lm(formula = .outcome ~ ., data = dat)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -1.13375 -0.36778  0.06064  0.36578  0.96057
#
# Coefficients:
#              Estimate Std. Error t value Pr(>|t|)
# (Intercept) -1.648629   0.226603  -7.275 2.55e-11 ***
```

```
# waiting      0.072211   0.003136  23.026  < 2e-16 ***
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#
# Residual standard error: 0.4941 on 135 degrees of freedom
# Multiple R-squared:  0.7971,   Adjusted R-squared:  0.7956
# F-statistic: 530.2 on 1 and 135 DF,  p-value: < 2.2e-16
```

Таке ж завдання можна виконати за допомогою пакету caret. Хоча раніше це демонструвалося вручну, пакет caret пропонує зручну альтернативу. Функція train в пакеті caret полегшує побудову моделі. Вихідна змінна представлена eruptions, а waiting служить предиктором, розділені тильдою. Вказано призначений набір даних для побудови моделі, а параметр method встановлено на лінійне моделювання. Підсумок остаточної підгонки моделі, доступний з об’єкта modFit, створеного train, розкриває специфіку кінцевої моделі, яка використовується для прогнозування. Подібно до моделі, встановленої вручну, перетин становить −1,64, а коефіцієнт часу очікування становить 0,07.

7. Висновок

У цій лекції ви дізналися про пакет caret, інструменти для створення властивостей і попередню обробку даних.