Лекція 15. Прогнозування за допомогою дерев, випадкових лісів і прогнозування на основі моделі

У цій лекції ви зосередитеся на дослідженні різних алгоритмів машинного навчання.

Мета заняття:

- реалізувати алгоритми на основі дерев для прогнозного моделювання, розуміючи їх принципи та застосування;
- застосувати техніку беггінгу, щоб покращити продуктивність моделі та зменшити перепідгонку;
- використовувати випадкові ліси та методи підсилення для подальшого підвищення точності прогнозування та надійності.

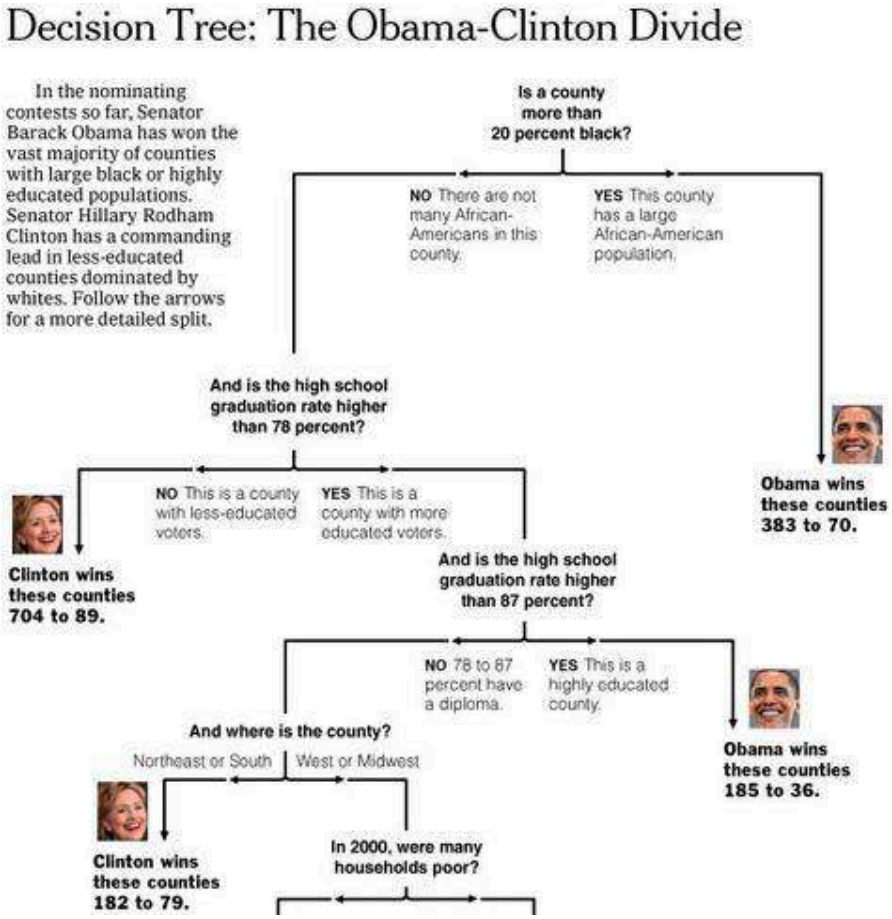
Зміст:

1. Прогнозування з деревами

Основна ідея прогнозування за допомогою дерев передбачає використання набору змінних для прогнозування результату. Кожна змінна використовується для розподілу результату на різні групи. Потім оцінюється однорідність результату в кожній групі. Цей процес триває, у разі потреби розподіляючи результати, доки групи не стануть достатньо однорідними або малими. Переваги цього підходу включають легкість інтерпретації та кращу продуктивність у нелінійних параметрах порівняно з моделями лінійної регресії. Однак без запобіжних заходів, таких як перехресна перевірка, це може призвести до перепідгонки. Крім того, оцінка невизначеності може бути більш складною порівняно з моделями лінійної регресії. Загалом результати можуть відрізнятися залежно від конкретних зібраних параметрів або змінних.

Ось приклад того, як виглядає дерево рішень після його створення. Це дерево рішень було представлено в New York Times під час виборів 2008 року, коли Барак Обама змагався з Гілларі Клінтон за номінацію від Демократичної партії на пост президента. Модель прогнозу була створена, щоб ставити запитання кожній із різних

змінних у наборі даних. Найкращий розподіл відбувся зі змінною, яка вказувала, чи було в окрузі більше 20% афроамериканців. Якщо так, то в окрузі було набагато більше шансів проголосувати за Барака Обаму; інакше було б більше шансів проголосувати за Гілларі Клінтон. Потім алгоритм шукав інші змінні, щоб розділити ці підгрупи далі, поки не досягне найменших розглянутих підгруп. В межах кожного листка дерева прогнози були досить однорідними. Наприклад, Обама виграв 383 з приблизно 450 округів, а Гілларі Клінтон виграла 704 з приблизно 790 або 800 округів. Ці запитання фактично розділили округи на групи з однорідними прогнозами щодо перемоги кожного кандидата на виборах.



Основний алгоритм побудови одного з цих дерев полягає в тому, щоб почати з усіх змінних в одній великій групі. Потім знайдіть першу змінну, яка найкраще розбиває результати на дві різні однорідні групи. Потім дані поділяються на дві групи, які називаються листами, а щойно виконаний поділ називається вузлом. У межах кожного поділу алгоритм знову шукає всі змінні, включаючи змінну, яку щойно розділили, щоб знайти іншу змінну або поділ, які розділяють результат на ще більш однорідні групи. Цей процес триває, доки групи не стануть занадто малими або достатньо однорідними, щоб зупинити алгоритм.

Існують різні показники домішок, усі вони базуються на ймовірності, оціненій у конкретній групі. У листках всього n об'єктів, і можна порахувати, скільки разів певний клас з'являється в цьому листку, позначається як ймовірність $\widehat{p}$ для m-го листка та k-го класу: $\widehat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \, in \, Leaf \, m} \mathbb{1}\left(y_i = k\right)$. Помилка неправильної класифікації дорівнює 1 мінус ймовірність відповідності найпоширенішому класу в цьому листку. Наприклад, якщо більшість округів проголосують за Барака Обаму у листку, тоді помилка неправильної класифікації дорівнює 1 мінус ймовірність голосування за нього: $1 - \widehat{p}_{mk(m)}$. 0 вказує на ідеальну чистоту, що означає відсутність помилок класифікації, тоді як 0,5 означає відсутність чистоти, що вказує на збалансований листок між двома результатами, що призводить до відсутності однорідності в цьому листку. Подібним чином існує те, що називається індексом Джіні, що відрізняється від коефіцієнта Джіні в економіці. По суті, це 1 мінус сума квадратів ймовірностей належності до будь-якого з різних класів: $\sum_{k \neq k'} \widehat{p}_{mk} \times \widehat{p}_{mk'} = \sum_{k=1}^{K} \widehat{p}_{mk}\left(1 - \widehat{p}_{mk}\right) = 1 - \sum_{k=1}^{K} p_{mk}^2$. Знову ж таки, значення 0 вказує на ідеальну чистоту, де один клас має ймовірність 1, а всі інші класи мають ймовірність 0. Значення 0,5 вказує на відсутність чистоти, тобто всі класи ідеально збалансовані в кожному листку.

Відхилення і приріст інформації є іншими мірами, які можна використовувати: $-\sum_{k=1}^{K} \widehat{p}_{mk} log_2 p_{mk}$. Відхилення обчислюється за допомогою натурального логарифма (логарифм з основою e), тоді як приріст інформації використовує логарифм з основою 2. По суті, це від'ємна сума ймовірності бути віднесеним до класу k у листку m, помножена на логарифм цієї ймовірності. Значення 0 вказує на ідеальну чистоту в листку, а значення 1 вказує на відсутність чистоти.

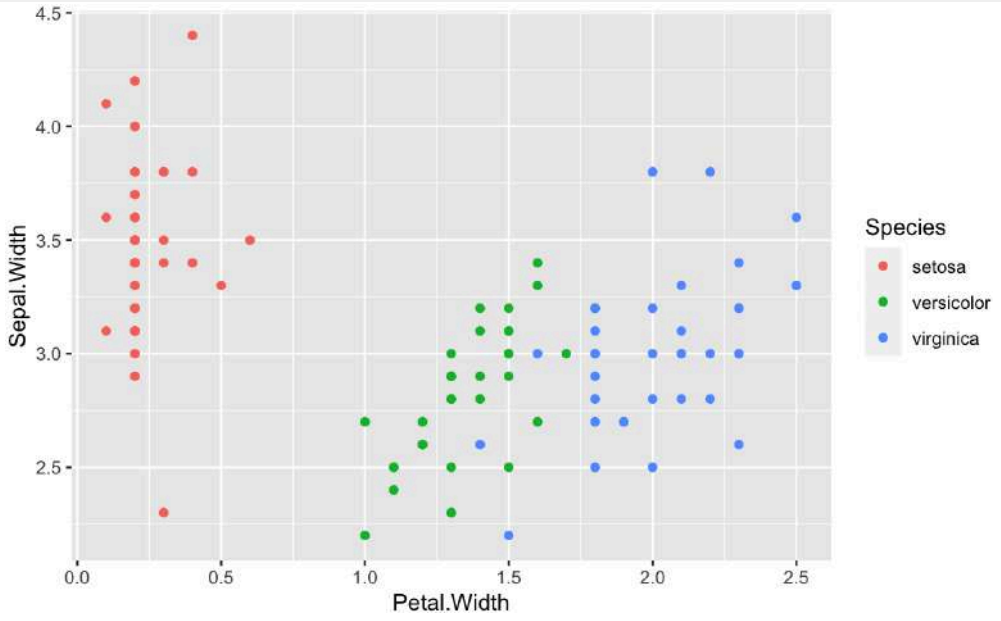
Ось як вони виглядають на конкретному прикладі. Припустимо, що є змінна, яка намагається розділити точки на сині та червоні. Якщо розділення призводить до 15 синіх точок і лише однієї червоної точки у листку, це відносно чисто. Коефіцієнт неправильної класифікації буде 1/16, що вказує на низьке значення. Індекс Джіні також буде низьким, що відображає чистоту. Подібним чином приріст інформації буде прагнути до 0. Тепер розглянемо розподіл, де половина значень є синіми, а половина – червоними. Це був би невдалий розподіл, тому що це як підкидання монети між

класами в цьому листку. У цьому сценарії рівень неправильної класифікації, індекс Джіні та приріст інформації будуть високими, що вказуватиме на погане розділення двох груп.

```
data(iris)
library(ggplot2)
names(iris)
# [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
table(iris$Species)
#     setosa versicolor  virginica
#         50         50         50
inTrain <- createDataPartition(y=iris$Species, p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training)
# [1] 105   5
dim(testing)
# [1] 45  5
```

Давайте розглянемо приклад із використанням набору даних iris. Цей набір даних досить старий, але добре проілюструє концепцію. Спочатку дані завантажуються за допомогою команди data(iris). Потім завантажується бібліотека ggplot2 для створення графіків. Набір даних містить такі змінні, як Sepal.Length, Sepal.Width, Petal.Length, та Petal.Width, причому Species є цільовою змінною. Існує 50 прикладів трьох різних видів, які прогнозуються за допомогою цих змінних. Як зазвичай, дані розділені на навчальний набір і тестовий набір.

```
qplot(Petal.Width, Sepal.Width, colour=Species, data=training)
```



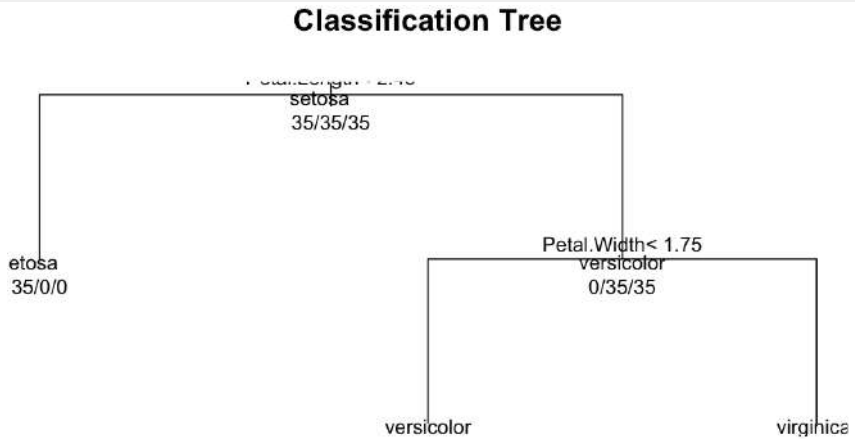
Першим кроком є побудова графіка Petal.Width і Sepal.Width. Вісь x представляє ширину пелюстки, а вісь y – ширину чашолистка, і кожна точка забарвлена відповідно до свого виду. Видно три чіткі кластери, що вказує на проблему

класифікації, яку легко розділити. Хоча лінійна модель може вважати це складним завданням, більш просунуті дерева класифікації повинні добре впоратися з цим.

```
library(caret)
modFit <- train(Species ~ ., method="rpart", data=training)
print(modFit$finalModel)
# n= 105
#
# node), split, n, loss, yval, (yprob)
#       * denotes terminal node
#
# 1) root 105 70 setosa (0.33333333 0.33333333 0.33333333)
#   2) Petal.Length< 2.45 35  0 setosa (1.00000000 0.00000000 0.00000000) *
#   3) Petal.Length>=2.45 70 35 versicolor (0.00000000 0.50000000 0.50000000)
#     6) Petal.Width< 1.75 37  3 versicolor (0.00000000 0.91891892 0.08108108) *
#     7) Petal.Width>=1.75 33  1 virginica (0.00000000 0.03030303 0.96969697) *
```

Модель можна підігнати за допомогою функції train в caret. Змінною результату є Species, а всі інші змінні використовуються для прогнозування, позначаються тильдою та крапкою. Вказаний метод – rpart, який є пакетом для дерев регресії та класифікації. Для цього процесу використовуються навчальні дані. Вивчаючи остаточну модель, можна спостерігати за вузлами, їх розбиттям і ймовірність належності до кожного класу для кожного розбиття. Наприклад, умова розділення як “Petal.Length < 2.45” вказує на те, що всі приклади з довжиною пелюстки менше ніж 2.45 класифікуються як setosa. Ці розбиття моделі дають змогу зрозуміти поведінку класифікаційного дерева.

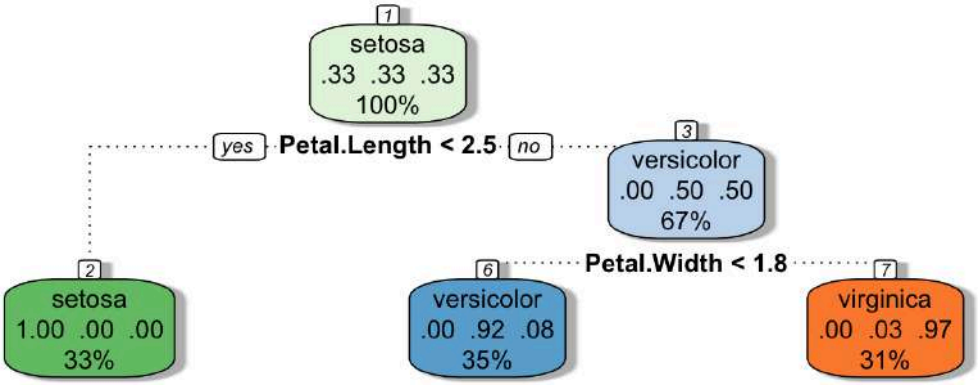
```
plot(modFit$finalModel, uniform=TRUE, main="Classification Tree")
text(modFit$finalModel, use.n=TRUE, all=TRUE, cex=.8)
```



Ви можете створити діаграму дерева класифікації, просто зобразивши на графіку finalModel. Це генерує дендрограму, як показано тут. Хоча вона може виглядати трохи зрізаною, структура помітна. Наприклад, якщо довжина пелюстки менше 2,45, призначенням є setosa. Проводячи ліворуч або праворуч від цього вузла, можна спостерігати результат класифікації на основі того, відповідає довжина пелюстки

цьому критерію чи ні. Подальші розподіли можна простежити аналогічно, щоб визначити загальну класифікацію для будь-якого прикладу.

```
library(rattle)
fancyRpartPlot(modFit$finalModel)
```



Більш візуально привабливу версію графіка можна створити за допомогою пакета rattle. Використовуючи функцію fancyRpartPlot і надаючи їй finalModel, підігнану за допомогою caret, створюється більш естетично приваблива дендрограма. На цьому вдосконаленому графіку стає зрозуміліше, що якщо довжина пелюстки менше 2,5, шлях веде ліворуч, тоді як якщо вона перевищує 2,5, шлях повертає праворуч. Далі в межах цього поділу, якщо ширина пелюстки менше 1,8, траєкторія йде вниз ліворуч, а якщо вона перевищує цей поріг, вона продовжується праворуч. Ця покращена візуалізація сприяє кращому розумінню процесу класифікації.

```
predict(modFit, newData=testing)
```

Нові значення можна прогнозувати за допомогою функції predict, подібно до інших моделей лінійної регресії. У цьому випадку прогнозування передбачає певну мітку класу, оскільки дерево класифікації було створено для цієї мети. Коли нові дані, відомі як дані тестування, надаються в модель, вона створює прогнози для різних категорій видів, оскільки прогнозує клас для кожної змінної.

Дерева класифікації – це нелінійні моделі, які за своєю суттю враховують взаємодію між змінними, що є важливим аспектом, про який слід пам’ятати. Ця модель ґрунтується на зв’язках між декількома змінними, і, маючи справу з численними класами для змінної, існує ризик перепідгонки. Перетворення даних можуть мати меншу значущість; монотонні перетворення, які змінюють шкалу, але зберігають порядок значень, дають ідентичні розбиття даних у класифікаційних або регресійних деревах. Ці дерева також можуть вирішувати проблеми регресії, причому середньоквадратична помилка служить альтернативною мірою домішки. Доступні різні

варіанти побудови дерева, включно з пакетом caret, за допомогою пакета party, пакета rpart або пакета tree, який пропонує додаткові функції для побудови моделі.

2. Беггінг

Беггінг (англ. bagging), абревіатура від агрегування бутстрапу (англ. bootstrap aggregating), передбачає усереднення кількох моделей для досягнення більш плавної підгонки моделі. Цей підхід врівноважує потенційне зміщення та дисперсію підгонки складних моделей.

Агрегування бутстрапу передбачає повторну вибірку набору даних для створення кількох версій даних. Цей процес подібний до бутстрепінгу, концепції, яка розглядалася раніше. Після створення цих повторно вибраних наборів даних із заміною функція прогнозування перераховується для кожного набору даних. Прогнози з цих моделей потім усереднюються або голосуються більшістю для задач класифікації. У той час як зміщення залишається таким же, як і в окремих моделях, мінливість зменшується завдяки усередненню кількох предикторів. Беггінг особливо ефективний для нелінійних функцій.

```
library(ElemStatLearn)
data(ozone, package="ElemStatLearn")
ozone <- ozone[order(ozone$ozone),]
head(ozone)
#     ozone radiation temperature wind
# 17      1         8          59  9.7
# 19      4        25          61  9.7
# 14      6        78          57 18.4
# 45      7        48          80 14.3
# 106     7        49          69 10.3
# 7       8        19          61 20.1
```

Повертаючись до даних ozone, які знаходяться в пакеті ElemStatLearn, набір даних завантажується. Згодом набір даних упорядковується на основі змінної результату ozone для демонстраційних цілей. Під час перевірки набору даних ідентифіковано чотири змінні: ozone, radiation, temperature та wind. Метою є прогнозування temperature як функції від ozone.

```
ll <- matrix(NA, nrow=10, ncol=155)
for (i in 1:10) {
  ss <- sample(1:dim(ozone)[1], replace=T)
  ozone0 <- ozone[ss,]
  ozone0 <- ozone0[order(ozone0$ozone),]
  loess0 <- loess (temperature ~ ozone, data=ozone0, span=0.2)
  ll[i,] <- predict(loess0, newdata=data.frame(ozone=1:155))
}
```

Щоб проілюструвати, як працює цей процес, наведемо приклад. Спочатку створюється матриця з 10 рядків і 155 стовпців. Згодом набір даних повторюється

десять разів за допомогою циклу. У кожній ітерації вибірка із заміною витягується з усього набору даних, утворюючи новий набір даних під назвою ozone0, що відповідає випадковій вибірці. Потім набір даних змінюється на основі змінної ozone. Крива loess підганяється для кожного набору даних із повторною вибіркою. loess – це гладка крива, яка використовується для моделювання зв'язку між змінними temperature та ozone, temperature служить результатом, тоді як ozone діє як предиктор. Підгонка кривої виконується за допомогою повторного набору даних із загальним параметром діапазону, що визначає плавність підгонки. Для кожної окремої кривої loess результат прогнозується для нового набору даних з використанням тих самих значень ozone в діапазоні від 1 до 155. Отже, і-й рядок результуючого об'єкта представляє прогноз, отриманий на основі кривої loess, що відповідає і-й повторній вибірці набору даних ozone. Процес передбачає повторну вибірку набору даних десять разів, підгонку плавної кривої через нього в кожній ітерації та подальше усереднення прогнозованих значень.

```
plot(ozone$ozone, ozone$temperature, pch=19, cex=0.5)
for (i in 1:10) {
  lines(1:155, ll[1,], col="grey", lwd=2)
}
lines(1:155, apply(ll, 2, mean), col="red", lwd=2)
```

На графіку вісь x представляє спостережувані значення озону, а вісь y – спостережувані значення температури, причому кожна чорна точка вказує на окреме спостереження. Сірі лінії відповідають підгонці, отриманій з кожного повторно відібраного набору даних, фіксуючи значну кількість мінливості, але, можливо, демонструючи надмірну кривизну. Однак після усереднення цих ліній з'являється більш гладка крива, ближча до центральної тенденції набору даних, зображена червоною лінією. Ця червона лінія позначає криву loess, яка, по суті, є середнім значенням кількох підігнаних кривих loess, отриманих із повторно відібраних наборів даних.

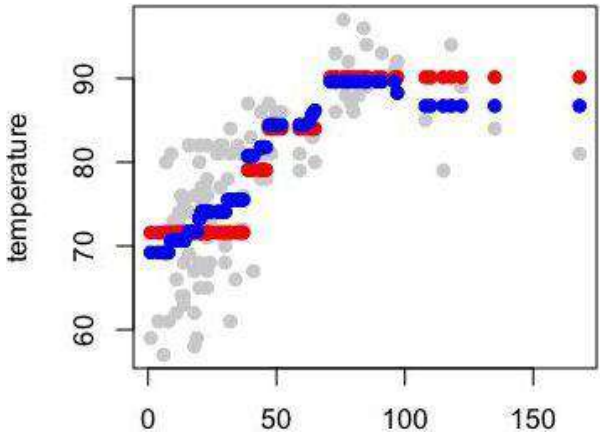
Є докази, які демонструють, що оцінка беггінгу постійно виявляє знижену мінливість, зберігаючи порівнянну похибку з індивідуальними підгонками моделі. У пакеті caret існують певні моделі, які за своєю суттю включають беггінг. Під час використання функції train можна вказати метод як bagEarth, treebag або bagFDA, які є спеціальними моделями беггінгу, які надаються пакетом caret.

```
predictors = data.frame(ozone=ozone$ozone)
temperature = ozone$temperature
treebag <- bag(predictors, temperature, B = 10,
```

```
            bagControl = bagControl(fit = ctreeBag$fit,
                                    predict = ctreeBag$pred,
                                    aggregate = ctreeBag$aggregate))
```

Крім того, ви можете розробити спеціальну функцію беггінгу в пакеті caret. Це передбачає більш просунутий підхід, тому бажано уважно переглянути документацію, перш ніж намагатися її реалізувати. Основна концепція передбачає організацію змінної предиктора у фреймі даних, де предиктори складають фрейм даних, що містить дані ozone. Крім того, вказується змінна результату, наприклад змінна temperature з набору даних. Потім це налаштування передається до функції bag у пакеті caret, де визначаються такі параметри, як предиктори, результат, кількість реплікацій і підвибірки. Крім того, параметр bagControl визначає процес підгонки моделі. Це включає визначення функції підгонки, методу прогнозування та підходу до агрегації для консолідації прогнозів у відтворених зразках.

```
plot(ozone$ozone, temperature, col='lightgrey', pch=19)
points(ozone$ozone, predict(treebag$fits[[1]]$fit, predictors), pch=19,
col="red")
points(ozone$ozone, predict(treebag, predictors), pch=19, col="blue")
```



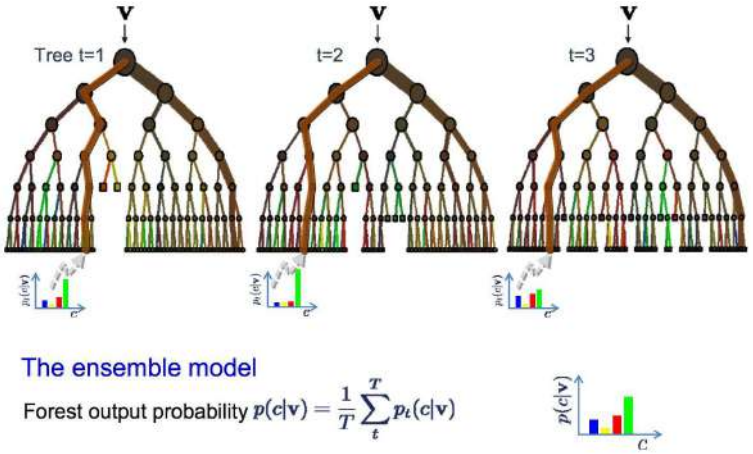
Вивчаючи налаштовану bag версію умовних регресійних дерев, ви можете помітити деякі переваги, продемонстровані раніше з беггінговим loess. На зображенні показано залежність озону на осі x від температури на осі y, а невеликі сірі точки позначають фактичні спостережувані значення. Червоні точки означають підгонку з одного дерева умовної регресії. Стає очевидним, що, наприклад, червона лінія залишається плоскою і не в змозі вловити основну тенденцію, очевидну в точках даних нижче. Однак після усереднення понад десяти різних підгонок беггінгової моделі з використанням цих умовних регресійних дерев, збільшення значень стає помітним у синій підгонці, що представляє підгонку від беггінгової регресії.

Беггінг є дуже вигідним для нелінійних моделей і широко використовується, особливо з деревами. Розширенням цієї концепції є випадковий ліс. Кілька моделей інтегрують беггінг в основну функцію train caret, як згадувалося раніше. Крім того, можна розробити власні функції беггінгу, адаптовані до будь-якого алгоритму класифікації або прогнозування, який цікавить. Однак важливо запам'ятати фундаментальний принцип: перевибірка даних, перенавчання нелінійної моделі, а потім агрегування підгонок моделі через повторні вибірки для досягнення більш плавної підгонки моделі порівняно з окремими підгонками.

3. Випадкові ліси

Випадкові ліси можна концептуалізувати як розширення беггінгу для дерев класифікації та регресії. Основна концепція віддзеркалює концепцію беггінгу: бутстрепінг вибірки застосовується до спостережуваних даних, генеруючи кілька наборів навчальних даних із повторною вибіркою, на основі яких реконструюються дерева класифікації або регресії. Однак ключова відмінність полягає в процесі вибору змінних під час кожного розділення дерев. Тут розглядається лише підмножина змінних, яка підлягає бутстрепінгу при кожному розділенні. Це урізноманітнює набір потенційних дерев, які можна створити. Основна стратегія передбачає вирощування значної кількості дерев, які згодом об'єднуються шляхом голосування або усереднення для отримання прогнозів щодо нових результатів.

Цей підхід має високу точність і широко використовується разом із підсиленням як один із найефективніших методів прогнозування. Тим не менш, він має недоліки. Випадкові ліси можуть бути обчислювально інтенсивними через необхідність побудови багатьох дерев. Крім того, їх інтерпретація є складною, враховуючи агрегацію безлічі дерев, що представляють бутстрепінг вибірки зі складними вузлами бутстрепінгу. Крім того, існує ризик перепідгонки, що посилюється труднощами визначення того, які дерева сприяють цій проблемі. Тому використання перехресної перевірки має вирішальне значення при побудові випадкових лісів.



Ось ілюстрація того, як випадкові ліси працюють на практиці. Процес передбачає побудову багатьох дерев, кожне з яких базується на бутстрепінгу вибірки, отриманої з набору даних. Наприклад, одне дерево будується на одній випадковій підвибірці, інше – на окремій підвибірці тощо. У кожному вузлі кожного дерева розглядається унікальна підмножина змінних для потенційного розділення. Коли вводиться нове спостереження, таке як спостереження V, зображене тут, воно проходить через кожне дерево послідовно. У цьому прикладі спостереження переміщається по першому дереву, закінчуючи певним вузлом листка, де отримує відповідний прогноз. Ця послідовність повторюється для наступних дерев, що призводить до дещо інших прогнозів на кожному кроці. Нарешті, прогнози з усіх дерев усереднюються, щоб отримати прогнозні ймовірності для кожного класу в усьому ансамблі дерев.

```
data(iris)
library(caret)
library(ggplot2)
inTrain <- createDataPartition(y=iris$Species, p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
```

Ось приклад, який демонструє, як функціонують випадкові ліси. Буде завантажено набір даних iris, а для візуалізації графіків буде використано пакет ggplot2. Буде створено навчальний набір та тестовий набір, при цьому модель буде побудована виключно на навчальних даних.

```
modFit <- train(Species ~ ., data=training, method="rf", prox=TRUE)
modFit
# Random Forest
#
# 105 samples
#   4 predictor
#   3 classes: 'setosa', 'versicolor', 'virginica'
```

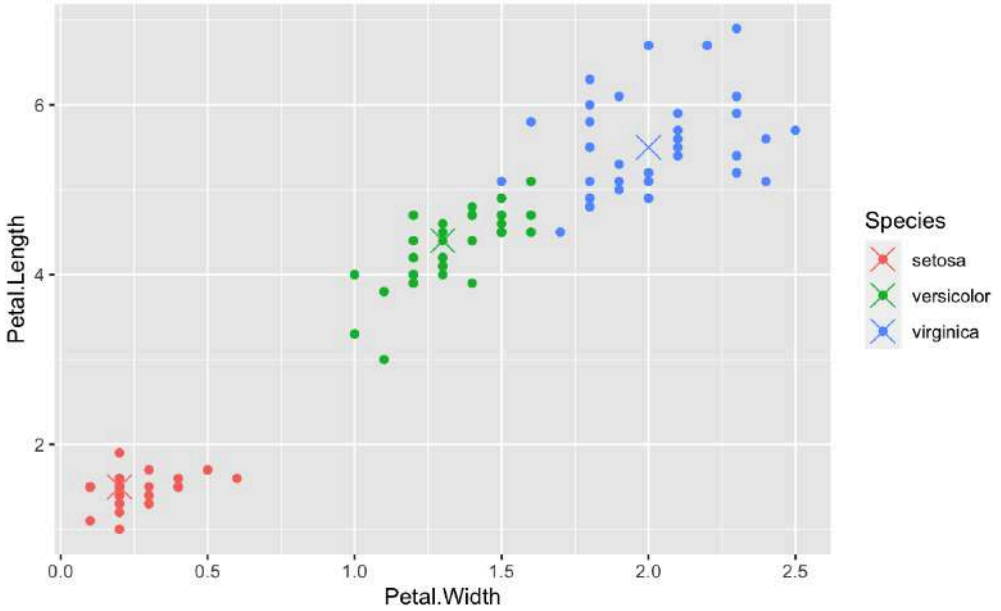
```
#
# No pre-processing
# Resampling: Bootstrapped (25 reps)
# Summary of sample sizes: 105, 105, 105, 105, 105, 105, ...
# Resampling results across tuning parameters:
#
#   mtry  Accuracy   Kappa
#   2     0.9472119  0.9199190
#   3     0.9460355  0.9181937
#   4     0.9427523  0.9131338
#
# Accuracy was used to select the optimal model using the largest value.
# The final value used for the model was mtry = 2.
```

У пакеті caret використовується функція train, схожа на інші процеси побудови моделі. Навчальний набір даних передається їй разом із визначенням методу як “rf” для випадкового лісу. Результатом є Species, тоді як будь-які інші прогнозні змінні розглядаються як потенційні предиктори. Встановлення параметра prox у значення True надає додаткову інформацію під час процесу підгонки моделі. Після побудови моделі виконується повторний бутстрепінг перевибірки, а потім пробуються різні параметри налаштування, включаючи кількість дерев, які потрібно побудувати.

```
library(randomForest)
getTree(modFit$finalModel, k=2)
#    left daughter right daughter split var split point status prediction
# 1              2              3         1        5.55      1          0
# 2              4              5         4        0.80      1          0
# 3              6              7         3        4.90      1          0
# 4              0              0         0        0.00     -1          1
# 5              8              9         3        4.45      1          0
# 6             10             11         4        1.65      1          0
# 7             12             13         3        5.15      1          0
# 8              0              0         0        0.00     -1          2
# 9              0              0         0        0.00     -1          3
# 10             0              0         0        0.00     -1          2
# 11            14             15         2        3.10      1          0
# 12            16             17         1        5.90      1          0
# 13             0              0         0        0.00     -1          3
# 14             0              0         0        0.00     -1          3
# 15             0              0         0        0.00     -1          2
# 16             0              0         0        0.00     -1          3
# 17            18             19         1        6.15      1          0
# 18             0              0         0        0.00     -1          2
# 19             0              0         0        0.00     -1          3
```

За допомогою функції getTree можна перевірити конкретне дерево в кінцевій моделі. Застосування getTree до кінцевої моделі та вказівка бажаного номера дерева відкриває структуру дерева. Кожен рядок відповідає певному розділенню, що вказує на лівий та правий дочірні елементи дерева, змінну, за допомогою якої відбувається розділення, значення розділення та прогноз, отриманий у результаті цього розділення.

```
irisP <- classCenter(training[,c(3,4)], training$Species,
modFit$finalModel$prox)
irisP <- as.data.frame(irisP)
irisP$Species <- rownames(irisP)
p <- qplot(Petal.Width, Petal.Length, col=Species, data=training)
p + geom_point(aes (x=Petal.Width, y=Petal.Length, col=Species), size=5,
shape=4, data=irisP)
```

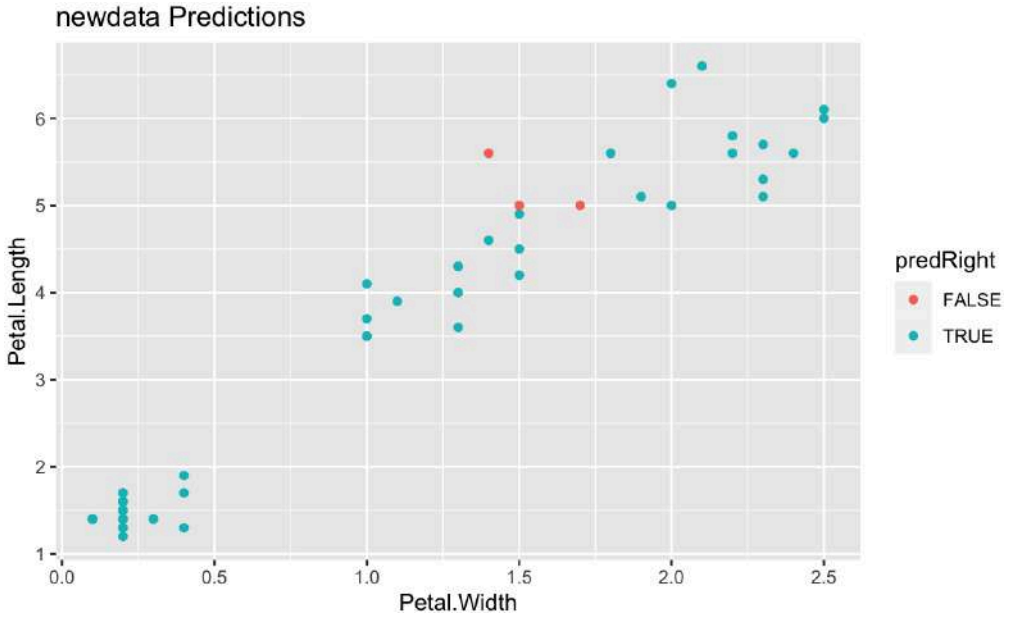


Ви можете використовувати інформацію про центри для спостереження за прогнозами або прогнозування центру класу. Тут розглядаються дві конкретні змінні Petal.Length і Petal.Width. Petal.Width відкладається на осі x, а Petal.Length – на осі y. Отримавши центри класу з підгонки моделі та надавши змінну prox, ви можете проаналізувати навчальний набір даних, щоб отримати ці центри. Згодом створюється набір даних із центрами та з видами. Буде створено графік залежності Petal.Width від Petal.Length, розфарбований видами в навчальному наборі даних. Потім додаються додаткові точки, що представляють Petal.Width і Petal.Length, причому кожна точка пофарбована відповідно до виду та отримана з центрів набору даних. Кожна точка на графіку представляє спостереження, а хрестики вказують на центр спостереження для кожного з різних прогнозів. Це показує, що кожен вид має прогноз для цих двох змінних, розташованих у центрі відповідної хмари точок для цього виду.

```
pred <- predict(modFit, testing)
testing$predRight <- pred==testing$Species
table(pred, testing$Species)
# pred         setosa versicolor virginica
#   setosa         15          0         0
#   versicolor      0         14         2
#   virginica       0          1        13
```

Ви можете прогнозувати нові значення за допомогою функції predict. Підгонка моделі та тестовий набір даних передаються для прогнозування. Крім того, встановлено змінну testing$predRight, яка вказує, чи відповідає прогноз виду в наборі даних тестування. Потім можна створити таблицю, що порівнює прогнози з фактичними видами, щоб оцінити ефективність моделі. Наприклад, у цьому прикладі модель випадкового лісу пропустила три значення, але в цілому вона продемонструвала високу точність своїх прогнозів.

```
qplot(Petal.Width, Petal.Length, colour=predRight, data=testing, main="newdata
Predictions")
```



Потім ви можете перевірити, які з трьох прогнозів були пропущені. Цікаво, що три пропущені точки, позначені червоним, потрапляють між двома різними класами. Один клас розташований у верхньому правому куті, а інший – посередині. Ці неправильно класифіковані точки лежать на межі між двома класами, надаючи розуміння областей як точних, так і неточних прогнозів.

Випадкові ліси, як правило, визначаються як один з найефективніших алгоритмів у конкурсах прогнозувань, поряд із підсиленням. Через велику кількість задіяних дерев їх може бути важко інтерпретувати. Однак вони забезпечують високу точність у різних проблемних областях. Функція rfcv може бути використана для забезпечення перехресної перевірки, хоча функція train в caret автоматизує цей процес.

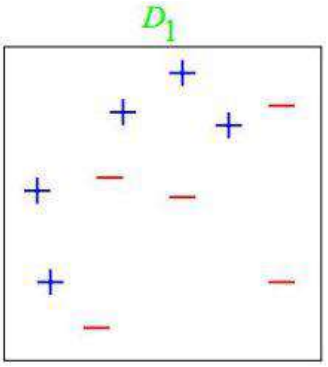
4. Підсилення

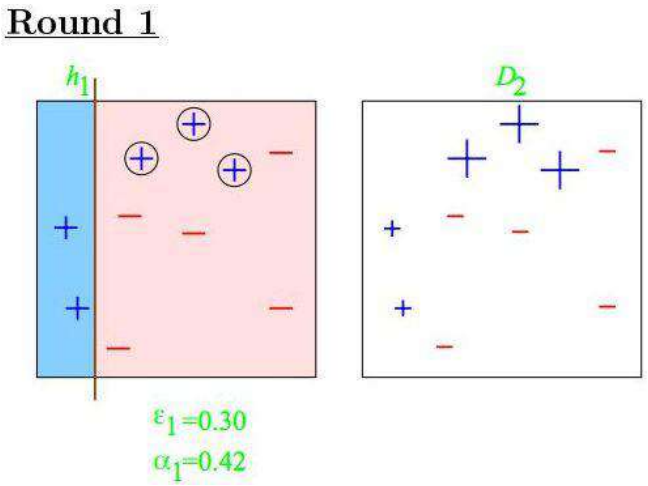
Підсилення, разом із випадковим лісом, є одними з найточніших доступних стандартних класифікаторів. Основна концепція передбачає використання великої

кількості потенційно слабких предикторів. Ці предиктори зважені, щоб отримати вигоду з їхніх індивідуальних сильних сторін, і узагальнені. Цей підхід нагадує методологію, що використовується в беггінгу для регресійних дерев або випадкового лісу, де численні класифікатори усереднюються, щоб отримати надійніший предиктор.

Основна ідея тут полягає у використанні k класифікаторів, які зазвичай беруться з одного класу класифікаторів. Приклади можуть включати всі потенційні класифікаційні дерева, регресійні моделі або різні відсікання, що ділять дані на сегменти. Потім створюється складений класифікатор шляхом поєднання та зважування цих функцій класифікації. Зважений ансамбль класифікаторів генерує прогнози для нових точок, позначених як $f(x)$, де α_t представляє вагу, помножену на класифікатор $h_t(x)$. Мета полягає в мінімізації помилок у навчальному наборі. Цей процес є ітераційним, з одним класифікатором h, вибраним на кожному кроці, ви обчислюєте ваги для наступного кроку на основі помилок, створених початковим h, підкреслюючи пропущені класифікації, вибираючи наступний етап і прогресуючи.

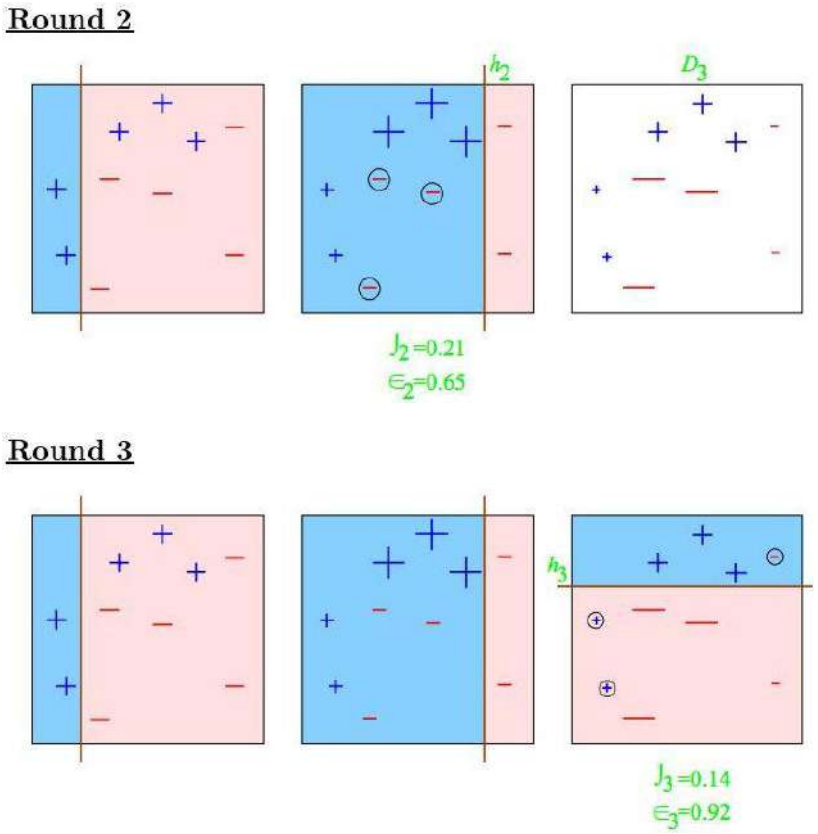
Ось простий приклад. Припустимо, що є спроба відрізнити сині знаки плюс і червоні знаки мінус за допомогою двох змінних предикторів. Перша змінна відкладається на вісь x, а друга змінна – на вісь y. Без конкретних назв, це служить базовою ілюстрацією. Мета полягає в тому, щоб розробити класифікатор, який ефективно розділяє ці змінні.





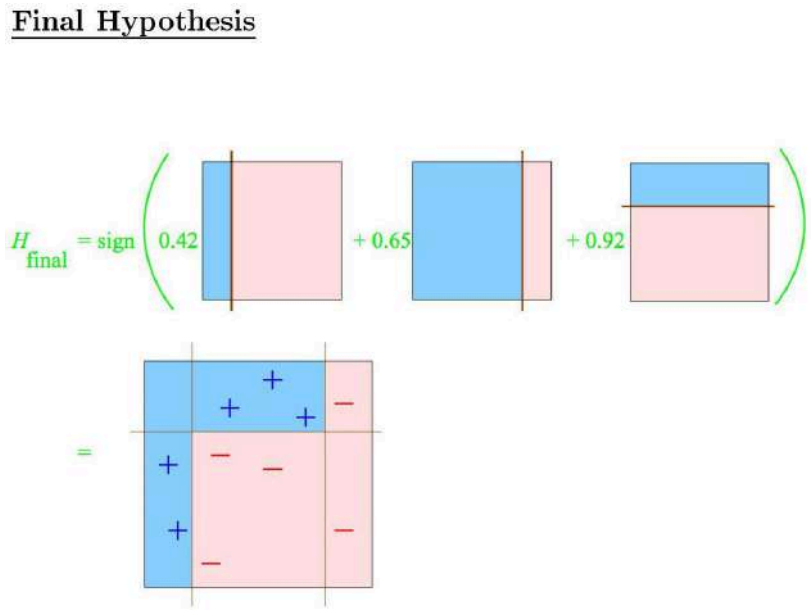
Концепція полягає в тому, чи може класифікатор ефективно розділяти ці змінні. Простий підхід може включати малювання вертикальної лінії та визначення

того, яка лінія ефективно розділяє ці точки. Ось приклад такого класифікатора: все, що знаходиться ліворуч від цієї вертикальної лінії, класифікується як синій плюс, тоді як усе, що знаходиться праворуч, класифікується як червоний мінус. Однак очевидно, що ви неправильно класифікували ці три точки у верхньому правому куті. Наступний крок включатиме створення цього класифікатора, обчислення рівня помилок (який у цьому випадку становить близько 30%), а потім підвищення ваги цих неправильно класифікованих точок. Ці збільшені точки представлені тут у більшому масштабі, що вказує на їх важливість для побудови наступного класифікатора.



Потім будується наступний класифікатор. У цьому сценарії другий класифікатор намалює вертикальну лінію в іншому місці. Таким чином, усе, що знаходиться праворуч від цієї лінії, класифікуватиметься як червоний мінус, а все, що ліворуч, буде синім плюсом. Знову три точки неправильно класифіковані, про що свідчить їх більший розмір для наступної ітерації. Рівень помилок перераховується для визначення вагових коефіцієнтів для наступного кроку. Згодом третій класифікатор має на меті правильно класифікувати раніше неправильно класифіковані точки. Наприклад, як плюси, так і мінуси в цих регіонах необхідно точно класифікувати. Щоб вирішити цю проблему, малюється горизонтальна лінія, де все, що знаходиться під нею, позначається червоним мінусом, а все, що знаходиться вище, – синім плюсом. Однак

неправильна класифікація відбувається з однією та двома точками в цій області. Отже, ці класифікатори зважені та комбіновані.



Підхід передбачає створення зваженої комбінації, що складається з 0,42 помножити на першу вертикальну лінію, 0,65 на другу вертикальну лінію та 0,92 на класифікацію з горизонтальної лінії. Коли ці правила класифікації поєднуються, отриманий класифікатор демонструє покращену продуктивність. Він успішно класифікує всі сині плюси і червоні мінуси. Незважаючи на те, що кожен окремий класифікатор є спрощеним – це просто лінія через площину, зважене агрегування в кінцевому підсумку створює надійний класифікатор.

```
library(ISLR)
data(Wage)
library(ggplot2)
library(caret)
Wage <- subset(Wage, select=-c(logwage))
inTrain <- createDataPartition(y=Wage$wage, p=0.7, list=FALSE)
training <- Wage[inTrain,]
testing <- Wage[-inTrain,]
```

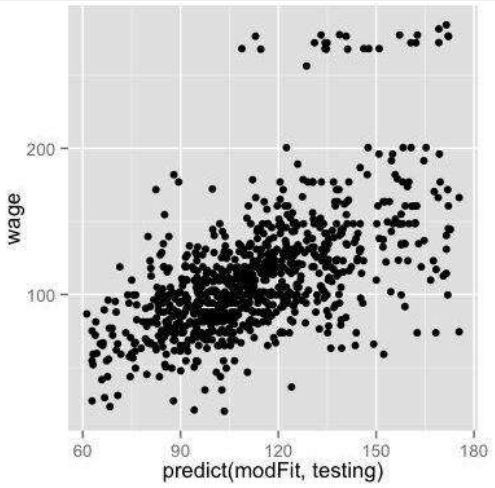
Тут приклад заробітної плати буде використано для демонстрації застосування алгоритму підсилення. Буде завантажено бібліотеку ISLR і дані Wage, а також бібліотеку ggplot2 і бібліотеку caret. Згодом буде згенеровано набір даних Wage без урахування змінної прогнозу, що цікавить, а саме змінної logwage Після цього буде створено набір для навчання та набір для тестування.

```
modFit <- train(wage ~ ., method="gbm" , data=training, verbose=FALSE)
print(modFit)
# Stochastic Gradient Boosting
#
```

```
# 2102 samples
#    9 predictor
#
# No pre-processing
# Resampling: Bootstrapped (25 reps)
# Summary of sample sizes: 2102, 2102, 2102, 2102, 2102, 2102, ...
# Resampling results across tuning parameters:
#
#   interaction.depth  n.trees  RMSE      Rsquared   MAE
#   1                   50      33.28631  0.2974125  23.23333
#   1                  100      32.83266  0.3071017  22.91145
#   1                  150      32.80651  0.3082926  22.92726
#   2                   50      32.83063  0.3081790  22.87161
#   2                  100      32.80308  0.3090635  22.88049
#   2                  150      32.89544  0.3067995  22.97708
#   3                   50      32.76221  0.3101231  22.82146
#   3                  100      32.90770  0.3059507  23.00573
#   3                  150      33.11256  0.3001051  23.18566
#
# Tuning parameter 'shrinkage' was held constant at a value of 0.1
# Tuning
#  parameter 'n.minobsinnode' was held constant at a value of 10
# RMSE was used to select the optimal model using the smallest value.
# The final values used for the model were n.trees = 50, interaction.depth =
#  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

Змінну wage можна змоделювати як функцію всіх решти змінних, позначених крапкою. Для цієї мети використовується пакет gbm, який використовує підсилення за допомогою дерев. Щоб уникнути надмірного виведення під час використання методу gbm, рекомендовано встановити параметр verbose на False. Після виведення підгонки моделі можна спостерігати різну кількість використовуваних дерев і глибин взаємодії, що разом сприяє побудові розширеної версії дерев регресії.

```
qplot(predict (modFit, testing), wage, data=testing)
```



Тут зображено на графіку прогнозовані результати тестових наборів. Підгонка моделі в R використовується для прогнозування тестового набору порівняно зі змінною wage в тестовому наборі. Прогноз виглядає досить точним, хоча певна мінливість все

ще очевидна. Фундаментальна концепція, що лежить в основі підгонки дерева підсилення або будь-якого алгоритму підсилення, полягає в агрегуванні слабких класифікаторів і поєднанні їх із ваговими коефіцієнтами для отримання більш надійного класифікатора.

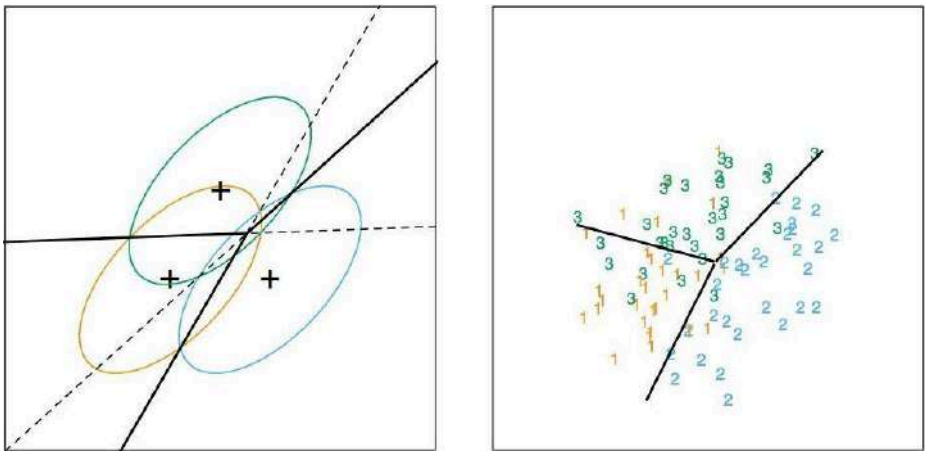
5. Прогнозування на основі моделі

Фундаментальна концепція прогнозування на основі моделі передбачає припущення, що дані відповідають певній ймовірнісній моделі. Використовуючи теорему Байєса, оптимальні класифікатори визначаються на основі цієї ймовірнісної моделі. Цей підхід пропонує перевагу використання будь-якої структури, яка може бути присутня у даних, наприклад, форму розподілу, що може призвести до обчислювальних зручностей. Крім того, це може дати прийнятну точність для реальних проблем, особливо тих, які узгоджуються з основним розподілом даних, змодельованим цілісно. Однак є недоліки, оскільки ці методи вимагають додаткових припущень щодо даних. Хоча ці припущення не обов'язково точно виконуються для ефективного функціонування алгоритмів прогнозування, значні відхилення можуть спричинити збій алгоритму. Крім того, неточності виникають, коли модель неправильна, що призводить до зниження точності прогнозу.

Ось підхід на основі моделі. Метою є побудова параметричної моделі або моделі на основі ймовірнісних розподілів. Для умовного розподілу ви зосереджуєтеся на ймовірності того, що результат Y дорівнює певному класу k за певного набору змінних предиктора X. Загальний підхід передбачає застосування теореми Байєса. По суті, ви прагнете зрозуміти ймовірність того, що Y дорівнює k, заданих спостережуваними змінними X, виражену як ймовірність того, що X дорівнює x, якщо Y дорівнює k, помножену на ймовірність того, що Y дорівнює k, поділену за законом повної ймовірності. Потім ви припускаєте параметричну модель для розподілу ознак, задану класом, позначеним як $P(x|y = k)$, і попереднім припущенням, що кожен елемент походить із певного класу, представленого π_k. Отже, ви можете змоделювати розподіл або ймовірність того, що y дорівнює k за певного набору змінних предикторів, у вигляді частки, що включає моделі для змінних X і попередньої ймовірності. Попередні ймовірності, π_k, зазвичай заздалегідь визначаються з даних. Загальним вибором для $f_k(x)$ є розподіл Гауса, який може бути багатовимірним, якщо є кілька змінних x. Тоді параметри μ_k і σ_k^2 можна оцінити з даних. Після того, як ці

параметри оцінені, стає можливим обчислити ймовірність того, що Y належить до будь-якого даного класу, спостерігаючи за змінними предиктора. Потім змінні класифікуються відповідно до класу з найвищою ймовірністю.

Різноманітні моделі використовують цей підхід. Серед них лінійний дискримінантний аналіз виділяється як один з найпопулярніших. Він припускає, що $f_k(x)$ дотримується багатовимірного розподілу Гауса, тобто властивості демонструють такий розподіл у кожному класі, і існує узгоджена коваріаційна матриця для всіх класів. Це фактично тягне за собою проведення ліній через коваріантний простір, звідси і назву лінійного дискримінантного аналізу. Квадратичний дискримінантний аналіз подібний до лінійного дискримінантного аналізу, але дозволяє використовувати різні коваріаційні матриці для кожного класу, в результаті чого через дані малюються квадратичні криві замість ліній. Прогнозування на основі моделі вміщує складніші версії коваріаційної матриці під час побудови моделі. Наївні байєсівські класифікатори припускають незалежність властивостей. По суті, вони припускають незалежність між змінними предикторів протягом усього процесу побудови моделі. Хоча це припущення може не відповідати дійсності, воно все одно може виявитися корисним для цілей прогнозування.



Ось як зазвичай виглядають межі прийняття рішень для цих типів моделей прогнозування. Розглянемо три окремі групи точок, які ви прагнете віднести до класу 1, класу 2 або класу 3. За допомогою двох змінних, які використовуються для класифікації вздовж осей x і y, ви підбираєте розподіл Гауса для кожного класу. Ось один розподіл Гаусса, інший і третій. Згодом ви малюєте лінії, де ймовірність переходить від вищої в одному класі до іншої. Це призводить до таких ліній. Якщо ви перебуваєте по цей бік лінії, ви будете класифіковані як клас 2. З цього боку ви будете класифіковані як клас 3, а якщо ви знаходитесь тут, ви будете класифіковані як клас 1. По суті, процес включає

підгонку розподілів Гауса до даних і їх використання для окреслення ліній, які призначають точки найвищим апостеріорним ймовірностям.

Загалом, у цьому контексті використовується дискримінантна функція. Основна ідея включає дискримінантну функцію, структуровану таким чином: $\delta_k(x) = x^T\sum^{-1}\mu_k - \frac{1}{2}\mu_k\sum^{-1}\mu_k + log(\mu_k)$, де μ_k представляє середнє значення класу k для всіх ознак. $\sum^{-1}$ позначає обернену коваріаційну матрицю для цього класу, яка є однаковою коваріаційною матрицею для всіх класів. Потім є лінійний член в x, предиктори, а також коваріаційна матриця та середнє значення. По суті, процес передбачає заміну нового значення даних у цю функцію. Згодом вибирається значення k, яке дає найбільше значення цієї конкретної дискримінантної функції, визначаючи клас. Як правило, ці параметри можна оцінити за допомогою оцінки максимальної правдоподібності.

Наївний байєсів класифікатор ще більше спрощує проблему. Припустимо, хтось має на меті змоделювати ймовірність того, що y належить до певного класу k, з урахуванням набору змінних предиктора. Теорема Байєса використовується для вираження ймовірності класу k, враховуючи ці спостережувані змінні, як попередню ймовірність перебування в класі k, помножену на ймовірність усіх цих ознак, задану класом k, поділену на константу. Отже, ця ймовірність пропорційна попередній ймовірності, помноженій на ймовірність ознак, заданих класом k. Вибір найбільшого значення цієї величини прирівнюється до вибору найбільшої ймовірності через те, що постійний член у знаменнику є однаковим для всіх ймовірностей. Подальша розбивка показує, що ймовірність ознак і змінної класу k дорівнює попередній ймовірності, помноженій на ймовірність X_1 заданого класу k, помноженій на ймовірність усіх інших змінних, крім X_1, залежних від X_i та $Y = k$. Цей процес триває до тих пір, поки кожна властивість не матиме відповідний член, усі з яких залежать від попередніх змінних. Це припущення припускає незалежність серед змінних предикторів, спрощуючи аргумент обумовлення. Отже, модель ґрунтується на “наївному” припущенні про незалежність характеристик, незважаючи на те, що це часто не відповідає дійсності. Незважаючи на це спрощення, наївний байєсів класифікатор залишається ефективним у різних

додатках, особливо при роботі з численними двійковими або категоріальними змінними, такими як класифікація тексту чи документа.

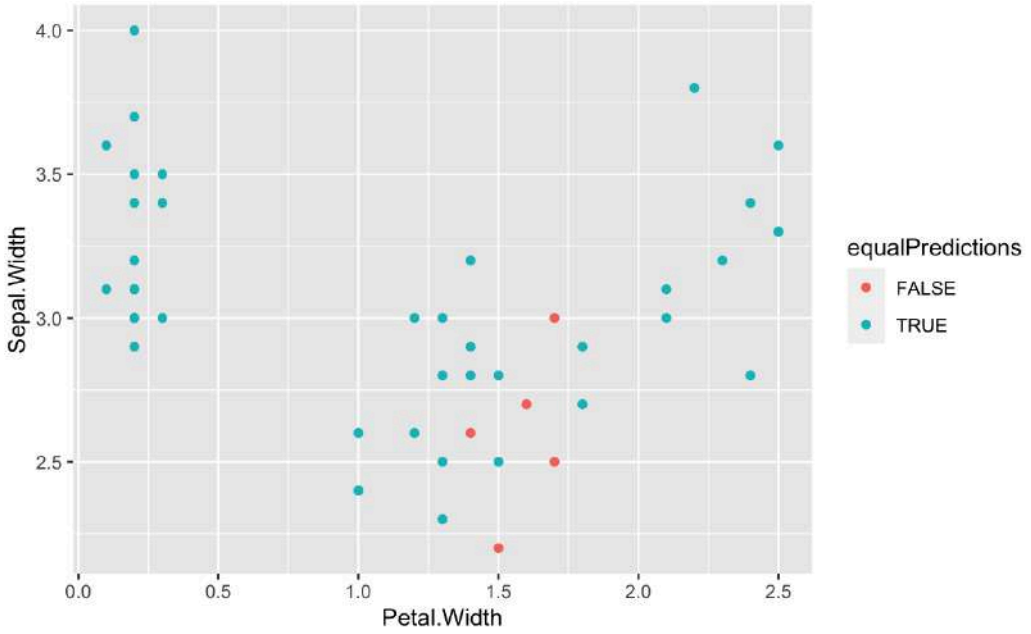
```
data(iris)
library(ggplot2)
names(iris)
# [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
table(iris$Species)
#     setosa versicolor  virginica
#         50         50         50
inTrain <- createDataPartition(y=iris$Species, p=0.7, list=FALSE)
training <- iris[inTrain,]
testing <- iris[-inTrain,]
dim(training)
# [1] 105   5
dim(testing)
# [1] 45  5
```

Щоб коротко проілюструвати, як функціонують ці два методи, ще раз буде використано дані iris. Завантажується бібліотека ggplot2 і визначаються змінні, які використовуються для прогнозування. Крім того, існує три різні види, які прогнозуються. Ці моделі можна підігнати на навчальному наборі та застосувати до тестового набору, подібно до інших алгоритмів прогнозування. Розділення створених даних знову використовується для створення окремих навчальних і тестових наборів.

```
modlda = train(Species ~ ., data=training, method="lda")
modnb = train(Species ~ ., data=training, method="nb")
plda = predict(modlda, testing)
pnb = predict(modnb, testing)
table(plda, pnb)
#             pnb
# plda         setosa versicolor virginica
#   setosa         15          0         0
#   versicolor      0         14         1
#   virginica       0          4        11
```

Модель lda може бути побудована на навчальному наборі за допомогою функції train з пакету caret. Модель будується шляхом передачі навчального набору та визначення методу як lda. Так само для наївної байєсової класифікації використовується метод nb. Потім можна зробити прогнози як на основі lda, так і наївної байєсової моделі на тестовому наборі, і можна створити таблицю прогнозів. Незважаючи на залежність між ознаками або предикторами, використання наївної байєсової класифікації дає правила прогнозування, дуже схожі на ті, що отримані з класифікації лінійного дискримінантного аналізу.

```
equalPredictions = (plda==pnb)
qplot(Petal.Width, Sepal.Width, colour=equalPredictions, data=testing)
```



Порівняння результатів показує, що лише декілька значень, розташованих між двома класами, здається по-різному класифікується двома результатами. Проте в цілому вони працюють дуже схоже.

6. Висновок

У цій лекції ви дізналися про різні алгоритми машинного навчання.