1.1 Дерева прийняття рішень (Decision Tree)

Дерева рішень *Decision Tree* є одним із найбільш ефективних інструментів інтелектуального аналізу даних та передбачуваної аналітики, які дозволяють вирішувати завдання *класифікації* та *регресії*.

Оскільки правила в деревах рішень виходять шляхом узагальнення безлічі окремих спостережень (*навчальних прикладів*), що описують *предметну область*, то за аналогією з відповідним методом логічного виведення їх називають індуктивними правилами, а процес *навчання* – індукцією дерев рішень.

У навчальній множині для прикладів має бути задане цільове значення, оскільки дерева рішень є моделями, що будуються на основі *навчання з учителем*. При цьому якщо цільова змінна дискретна (*мітка класу*), то модель називають деревом класифікації, а якщо безперервна, то деревом регресії.

1.1.1 Структура дерева рішень

Введемо до розгляду основні поняття, що використовуються в теорії дерев рішень.

Назва	Опис
Об'єкт	Приклад, шаблон, спостереження
Атрибут	Ознака, незалежна змінна, властивість
Цільова змінна	Залежна змінна, мітка класу
Вузол	Внутрішній вузол дерева, вузол перевірки
Кореневий вузол	Початковий вузол дерева рішень
Лист	Кінцевий вузол дерева, вузол рішення
Вирішальне правило	Умова у вузлі, перевірка

Власне, саме дерево рішень — метод подання *вирішальних правил* у ієрархічній структурі, що складається з елементів двох типів – вузлів (node) та листя (leaf). У вузлах знаходяться вирішальні правила і проводиться перевірка відповідності прикладів цього правила за яким-небудь *атрибутом* навчальної множини.

У найпростішому випадку, в результаті перевірки, множина прикладів, що потрапили у вузол, розбивається на дві підмножини, в одну з яких потрапляють приклади, що задовольняють правило, а

в інше не задовольняють. Потім до кожної підмножини знову застосовується правило і процедура рекурсивно повторюється доки буде досягнуто деяка умова зупинки алгоритму. В результаті в останньому вузлі перевірка та розбиття не проводиться і він оголошується листом. Лист визначає рішення для кожного прикладу, що в нього потрапив. Для дерева класифікації це *клас*, що асоціюється з вузлом, а для дерева регресії модальний інтервал цільової змінної, що відповідає листу.

Таким чином, на відміну від вузла, у листі міститься не правило, а підмножина об'єктів, що задовольняють всі правила гілки, яка закінчується цим листом. Очевидно, щоб потрапити в лист, приклад повинен задовольняти всі правила, що лежать на шляху до цього листа. Оскільки шлях у дереві до кожного листа єдиний, то й кожен приклад може потрапити лише в один лист, що забезпечує єдиність рішення.

1.1.2 Процес побудови

Процес побудови дерев рішень полягає в послідовному, рекурсивному розбиття навчальної множини на підмножини із застосуванням вирішальних правил у вузлах. Процес розбиття триває доти, доки всі вузли наприкінці всіх гілок не будуть оголошені листям. Оголошення вузла листом може статися природним чином (коли він міститиме єдиний об'єкт, або об'єкти лише одного класу), або після досягнення певної умови зупинки, що задається користувачем (наприклад, мінімально допустима кількість прикладів у вузлі або максимальна глибина дерева).

Алгоритми побудови дерев рішень відносять до категорії *жадібних алгоритмів*. Жадібними називаються алгоритми, які допускають, що локально-оптимальні рішення на кожному кроці (розбиття у вузлах), призводять до оптимального підсумкового рішення. У випадку дерев рішень це означає, що якщо один раз був обраний атрибут, і по ньому було розбито на підмножини, то алгоритм не може повернутися назад і вибрати інший атрибут, який дав би найкраще підсумкове розбиття. Тому на етапі побудови не можна сказати, чи забезпечить обраний атрибут, зрештою, оптимальне розбиття.

В основі більшості популярних алгоритмів навчання дерев рі-

шень лежить принцип "поділяй і володарюй". Алгоритмічно цей принцип реалізується в такий спосіб. Нехай задано навчальну множину X що містить n прикладів, для кожного з яких задано мітку класу $C_i (i = 1, \ldots, k)$, та m атрибутів $A_j (j = 1, \ldots, m)$ які, як передбачається, визначають належність об'єкта до того чи іншого класу. Тоді можливі три випадки:

1. Усі приклади множини X мають однакову мітку класу C_i (тобто всі навчальні приклади відносяться тільки до одного класу). Очевидно, що навчання в цьому випадку не має сенсу, оскільки всі приклади моделі, що пред'являються, будуть одного класу, який і "навчиться" розпізнавати модель. Саме дерево рішень у цьому випадку буде листом, асоційованим з класом C_i. Практичне використання такого дерева безглуздо, оскільки будь-який новий об'єкт воно відноситиме лише до цього класу.

2. Множина X взагалі не містить прикладів, тобто є порожньою множиною. У цьому випадку для нього теж буде створено лист (застосовувати правило, щоб створити вузол, до порожньої множини безглуздо), клас якого буде обраний з іншої множини (наприклад, клас, який найчастіше зустрічається в батьківській множині).

3. Множина X містить навчальні приклади всіх класів C_k. У цьому випадку потрібно розбити множину X на підмножини, асоційовані із класами. Для цього вибирається один із атрибутів A_j множини X який містить два і більше унікальних значень $(a_1, a_2, \ldots, a_p)$, де p – кількість унікальних значень ознаки. Потім множина X розбивається на p підмножин $(X_1, X_2, \ldots, X_p)$ кожна з яких включає приклади, що містять відповідне значення атрибута. Потім вибирається наступний атрибут і повторюється розбиття. Ця процедура рекурсивно повторюватиметься до тих пір, поки всі приклади в результуючих підмножинах не виявляться одного класу.

Побудова дерева рішень відбувається зверху донизу (від кореневого вузла до листя). Описана вище процедура є основою багатьох

сучасних алгоритмів побудови дерев рішень. Найбільш популярні алгоритми:

ID3 (Iterative Dichotomizer 3) – алгоритм дозволяє працювати тільки з дискретною цільовою змінною, тому дерева рішень, побудовані за допомогою даного алгоритму, є такими, що класифікують. Число нащадків у вузлі дерева не обмежене. Не може працювати із пропущеними даними.

C4.5 – удосконалена версія алгоритму ID3, до якої додано можливість роботи з пропущеними значеннями атрибутів.

CART (Classification and Regression Tree) – алгоритм навчання дерев рішень, що дозволяє використовувати як дискретну, так і безперервну цільову змінну, тобто вирішувати завдання класифікації, так і регресії. Алгоритм будує дерева, які в кожному вузлі мають лише два нащадки.

Основні етапи побудови

У ході побудови дерева рішень необхідно вирішити кілька основних проблем, з кожною з яких пов'язаний відповідний крок процесу навчання:

- Вибір атрибута, за яким буде проводитися розбиття в даному вузлі (атрибута розбиття).
- Вибір критерію зупинення навчання.
- Вибір методу відсікання гілок (спрощення).
- Оцінка точності збудованого дерева.

1.1.3 Вибір атрибуту розбиття

При формуванні правила для розбиття в черговому вузлі дерева необхідно вибрати атрибут, яким це буде зроблено. Загальне правило для цього можна сформулювати наступним чином: обраний атрибут повинен розбити безліч спостережень у вузлі так, щоб підмножини, що результують, містили приклади з однаковими мітками

класу, або були максимально наближені до цього, тобто. кількість об'єктів з інших класів ("домішок") у кожному з цих множин було якнайменше. Для цього було обрано різні критерії, найбільш популярними з яких стали теоретико-інформаційний та статистичний.

Теоретико-інформаційний критерій

Грунтується на поняттях теорії інформації, а саме – *інформаційної ентропії.*

$$S = -\sum_{i=1}^{n} \frac{N_i}{N} \log_2 \left(\frac{N_i}{N} \right) \tag{1.1}$$

де n – число класів у вихідній підмножині, N_i – кількість прикладів i-го класу, N – загальна кількість прикладів у підмножині. Таким чином, ентропія може розглядатися як міра неоднорідності підмножини за представленими у ньому класами. Коли класи представлені у рівних частках та невизначеність класифікації найбільша, ентропія також максимальна. Якщо приклади у вузлі ставляться одного класу, тобто $N = N_i$ логарифм від одиниці звертає ентропію в нуль.

Таким чином, найкращим атрибутом розбиття A_j буде той, який забезпечить максимальне зниження ентропії результуючої підмножини по відношенню до батьківської. На практиці, однак, говорять не про ентропію, а про величину, обернену їй, яка називається інформацією. Тоді найкращим атрибутом розбиття буде той, який забезпечить максимальний приріст інформації результуючого вузла щодо вихідного:

$$Gain(A) = S_0 - \sum_{i=1}^{n} \frac{N_i}{N} S_i, \tag{1.2}$$

де S_0 – інформаційна ентропія, пов'язана з підмножиною X до розбиття, S_i – інформаційна ентропія, пов'язана з підмножинами, отриманими після розбиття за атрибутом A. Таким чином, завдання вибору атрибуту розбиття у вузлі полягає у максимізації величини $Gain(A)$, що називається приростом інформації (від англ. Gain - приріст, збільшення). Тому сам теоретико-інформаційний підхід відомий як *критерій приросту інформації.* Він вперше був застосований в алгоритмі ID3, а потім у C4.5 та інших алгоритмах.

Статистичний підхід

В основі статистичного підходу лежить використання індексу Джіні (названий на честь італійського статистика та економіста Коррадо Джіні). Статистичний зміст цього показника в тому, що він показує, наскільки часто випадково обраний приклад навчальної множини буде розпізнаний неправильно, за умови, що цільові значення в цій множині були взяті з певного статистичного розподілу. Таким чином, індекс Джині фактично показує відстань між двома розподілами – розподілом цільових значень, і розподілом передбачень моделі. Очевидно, що чим менша ця відстань, тим краще працює модель.

Індекс Джіні може бути розрахований за формулою:

$$Gini(Q) = \sum_{i=1}^{n} p_i(1 - p_i), \tag{1.3}$$

де Q – результуюча множина, n – кількість класів в ньому, $p = N_i/N$ – імовірність і-го класу (виражена як відносна частота прикладів відповідного класу). Очевидно, що цей показник змінюється від 0 до 1. При цьому він дорівнює 0 якщо всі приклади Q відносяться до одного класу, і дорівнює 1, коли класи представлені в рівних пропорціях і є рівноймовірними. Тоді найкращим буде те розбиття, для якого значення індексу Джині буде мінімальним.

1.1.4 Критерій зупинення алгоритму

Теоретично, алгоритм навчання дерева рішень буде працювати доти, доки в результаті не будуть отримані абсолютно "чисті" підмножини , у кожному з яких будуть приклади одного класу. Правда, можливо, при цьому буде побудовано дерево, в якому для кожного прикладу буде створено окремий лист. Очевидно, що таке дерево виявиться марним, оскільки воно буде перенавченим — кожному прикладу відповідатиме свій унікальний шлях у дереві, а отже, і набір правил, актуальний лише для цього прикладу.

Перенавчання у випадку дерева рішень веде до тих самих наслідків, що й для нейронної мережі - точне розпізнавання прикладів, що беруть участь у навчанні та повна неспроможність нових

даних. Крім цього, перенавчені дерева мають дуже складну структуру і тому їх складно інтерпретувати. Очевидним вирішенням проблеми є примусова зупинка побудови дерева, доки воно не стало перенавченим. Для цього розроблено такі підходи.

Рання зупинка – алгоритм буде зупинено, як тільки буде досягнуто заданого значення деякого критерію, наприклад, процентної частки правильно розпізнаних прикладів. Єдиною перевагою підходу є зниження часу навчання. Головним недоліком є те, що рання зупинка завжди робиться на шкоду точності дерева, тому багато авторів рекомендують віддавати перевагу відсіканню гілок.

Обмеження глибини дерева – завдання максимальної кількості розбиття у гілках, після досягнення навчання зупиняється. Цей метод також веде до зниження точності дерева.

Завдання мінімально допустимого числа прикладів у вузлі – заборонити алгоритму створювати вузли з числом прикладів менше заданого (наприклад, 5). Це дозволить уникнути створення тривіальних розбиття і, відповідно, незначних правил.

Усі перелічені підходи є евристичними, тобто не гарантують кращого результату чи взагалі працюють лише у якихось окремих випадках. Тому до використання слід підходити з обережністю. Яких-небудь обґрунтованих рекомендацій щодо того, який метод краще працює, нині теж не існує. Тому аналітикам доводиться використовувати метод спроб та помилок.

1.1.5 Відсікання гілок

Як було зазначено вище, якщо "зростання" дерева не обмежити, то в результаті буде побудовано складне дерево з великою кількістю вузлів та листя. Як наслідок воно буде важко інтерпретованим. У той самий час вирішальні правила таких деревах, створюють вузли, у яких потрапляють два-три приклади, виявляються малозначними з практичної погляду.

Набагато краще мати дерево, що складається з малої кількості вузлів, яким відповідало б велику кількість прикладів з навчальної вибірки. Тому цікавий підхід, альтернативний ранній зупинці – побудувати всі можливі дерева і вибрати те з них, яке за розумної глибини забезпечує прийнятний рівень помилки розпізнавання, тобто знайти найбільш вигідний баланс між складністю та точністю дерева. Альтернативним підходом є так зване відсікання гілок (pruning). Він містить такі кроки:

1. Побудувати повне дерево (щоб усе листя містило приклади одного класу).

2. Визначити два показники: відносну точність моделі – відношення числа правильно розпізнаних прикладів до загального числа прикладів, та абсолютну помилку – число неправильно класифікованих прикладів.

3. Видалити з дерева листя і вузли, відсікання яких не призведе до значного зменшення точності моделі або збільшення помилки.

Відсікання гілок, очевидно, виробляється у напрямі, протилежному напрямку зростання дерева, тобто. знизу нагору, шляхом послідовного перетворення вузлів на листя. Перевагою відсікання гілок у порівнянні з ранньою зупинкою є можливість пошуку оптимального співвідношення між точністю та зрозумілістю дерева. Недоліком є більший час навчання через необхідність спочатку збудувати повне дерево.

1.1.6 Вилучення правил

Іноді навіть спрощене дерево рішень все ще є надто складним для візуального сприйняття та інтерпретації. У цьому випадку може виявитися корисним витягти з дерева вирішальні правила та організувати їх у набори, що описують класи.

Для отримання правил необхідно відстежити всі шляхи від кореневого вузла до листя дерева. Кожен такий шлях дасть правило, що складається з безлічі умов, що є перевіркою в кожному вузлі шляху.

Візуалізація складних дерев рішень як вирішальних правил замість ієрархічної структури з вузлів і листя може бути зручнішою для візуального сприйняття.

1.1.7 Постановка задачі

Провести класифікацію даних з використанням методу дерева рішень. Етапи розв'язання

1. Імпортувати вибірку для проведення навчання
2. Розділити всю вибірку на навчальну та тестову
3. Побудувати алгоритм навчання на навчальній вибірці
   (а) Порахувати ентропію за формулою (1.1) або показник Gini (1.3) для вихідної таблиці
   (б) Відсортувати таблицю за першою ознакою
   (в) Знайти значення ознаки для першого поділу таблиці, коли значення цільового вектору змінюється вперше
   (г) Порахувати приріст інформації при такому діленні за формулою (1.2)
   (д) Повторити процедуру для всіх можливих ділень за першою ознакою та за всіма ознаками і знайти оптимальне перше ділення таблиці для якого приріст інформації є максимальним.
   (е) Рекурсивно для кожного ділення повторити процедуру ділення
   (ж) Вихід з рекурсії здійснити за умови
      - перевищення фіксованої глибини дерева
      - мінімальної (фіксованої) кількості об'єктів у вузлі
      - мінімального (фіксованого) значення ентропії вузла
4. Перевірити точність роботи алгоритму на тестовій вибірці
5. Порівняти результати з Decision tree з sklearn
6. Оформити результати у вигляді звіту.

1.1.8 Приклад подання результату

У якості результатів роботи програми отримуємо словник з правилами рухів по дереву (датасет “iris”):
Мітка:

“ ” – корінь дерева

“l” – гілка вліво

“r” – гілка вправо

Елементи словника:

[0] – номер ознаки для ділення

[1] – номер рядка для ділення

[2] – значення ознаки для ділення

[3]:

none – не завершена гілка

Int(i) – клас до якого належить об’єкт

Типовий словник:
’’: [3, 70, 1.75, None], ’l’: [3, 69, 1.65, None], ’lr’: [None, None, None, 2], ’ll’: [3, 58, 1.45, None], ’lll’: [3, 32, 0.8, None], ’llll’: [None, None, None, 0], ’lllr’: [2, 24, 5.2, None], ’lllrl’: [None, None, None, 1], ’lllrr’: [None, None, None, 2], ’llr’: [2, 7, 4.95, None], ’llrl’: [None, None, None, 1], ’llrr’: [3, 1, 1.55, None], ’llrrl’: [None, None, None, 2], ’llrrr’: [None, None, None, 1], ’r’: [0, 3, 5.85, None], ’rl’: [None, None, None, 2], ’rr’: [1, 20, 3.15, None], ’rrl’: [None, None, None, 2], ’rrr’: [3, 0, 1.9, None], ’rrrl’: [None, None, None, 1], ’rrrr’: [None, None, None, 2]
Error in train = 1.0 Error in test = 0.933