Лекция 5. Научаване на множество от правила

В много случаи е удобно да научиш целевата функция във вид на множество от правила АКО – ТО, които заедно определят тази функция. В предишната лекция ние разгледахме един възможен начин за научаване на множество от правила – от начала се научава класификационното дърво, а след това то се превежда в еквивалентно множество от правила – по едно правило за всеки път от корена до едно от листата в дървото. В тази лекция ще разгледаме методи за директно научаване на правила.

5.1. Алгоритми за последователно покриване

Съществува голяма фамилия от алгоритми за научаване на правила, базиращи се върху стратегия, при която се научава само едно правило, изтриват се примери, които то покрива, след което процесът се повтаря отново. Подобни алгоритми се наричат *алгоритми за последователно покриване*. Да предположим, че имаме подпрограма НАУЧИ-ЕДНО-ПРАВИЛО, която приема като вход определено множество от положителни и отрицателни обучаващи примери, и извежда едно единствено правило, покриващо повечето от положителните примери и няколко от отрицателните. Ще изискваме това изходно правило да има висока точност, но не е задължително голям обсег на покриване. Под голямата точност разбираме, че предсказанията на правилото трябва да бъдат точни. Като допускаме възможността за малко покриване, това означава, че не изискваме от правилото да прави предсказания за всеки от обучаващите примери, а само за някое, може и доста ограничено, подмножество от обучаващи примери.

Разполагайки с подобна процедура за научаване само на едно правило, един очевиден подход към задачата за научаване на множество от правила е да извикаме процедурата НАУЧИ-ЕДНО-ПРАВИЛО върху всички обучаващи примери, да изтрием след това всички положителни обучаващи примери, покрити от новосъздаденото правило, а след това до извикаме същата процедура за научаване на второто правило и т.н. Процедурата се извиква толкова пъти, колко трябва, за да бъдат покрити с правила всички положителни обучаващи примери. Полученото на края дизюнктивно множество от правила може да бъде сортирано по такъв начин, че при опит за класифициране на новия пример по-точните правила ще се разглеждат по-рано. Един примерен алгоритъм за последователно покриване е представен в Табл. 5-1.

ПОСЛЕДОВАТЕЛНО-ПОКРИВАНЕ(*Цел_атрибут, Атрибути, Примери, Праг*)

- *Научени_правила* ← *{}*
- *Правило* ← НАУЧИ-ЕДНО-ПРАВИЛО(*Цел_атрибут, Атрибути, Примери*)
- **докато** ПОВЕДЕНИЕ(*Правило, Примери*) > *Праг* **направи**
  - *Научени_правила* ← *Научени правила* + *Правило*
  - *Примери* ← *Примери* – *{*примери, правилно класифицирани от *Правило}*

    - *Правило* ← НАУЧИ-ЕДНО-ПРАВИЛО(*Цел_атрибут, Атрибути, Примери*)
- *Научени_правила* ← **сортирай** *Научени_правила* съгласно ПОВЕДЕНИЕ върху *Примери*
- **върни** *Научени_правила*

Таблица 5-1. Алгоритъм за последователно покриване

Този алгоритъм за последователно покриване научава правила, докато става невъзможно да бъде научено правило, чието поведение е над задания *Праг*. ПОВЕДЕНИЕ е някоя зададена от потребителя процедура, оценяваща качеството на правило. Описаният алгоритъм е един от най-често използваните подходи към научаване на дизюнктивното множество от правила. Той свежда проблема за научаване на дизюнктивното множество от правила към последователността от по-лесни проблеми, всяка от която изисква научаване само на едно конюктивно правило. Тъй като задавайки една последователност от правила, алгоритмът осъществява евристичното търсене без възврат, той не може да гарантира намирането на най-малкото или най-доброто множество от правила, покриващи зададеното множество от обучаващите примери.

5.1.1. "Лъчево" (beam) търсене от общото към частното

Как трябва да бъде проектирана процедурата НАУЧИ-ЕДНО-ПРАВИЛО, за да отговаря на изискванията на алгоритъма за последователно покриване? Един ефективен подход към подобна имплементация е да организираме търсенето в пространството на хипотези по същия обобщен начин, както при алгоритъма ID3, но на всяка стъпка да следваме само по най-обещаващ клон на дървото. Както е показано на Фиг. 5-1, илюстрираща дървото на търсене, търсенето започва с разглеждането на най-общото правило – правило с празно предусловие: то покрива всички възможни примери. След това по евристичен начин към предусловието се добавя проверка на атрибут, която най-много подобрява поведението на правилото, измерено върху обучаващите примери. След като такъв тест се добавя към правилото, процесът се повтаря, евристично добавяйки следващата проверка на атрибута и т.н. Както и при ID3 този процес построява хипотезата чрез евристичното добавяне на теста върху атрибута, докато хипотезата не достигне приемливото ниво на поведение. В отличие от ID3 този начин на имплементация на процедурата за научаване на едно правило проследява на всяка стъпка само един възел-наследник – тази двойка атрибут – стойност, която води до най-доброто поведение, докато при ID3 се построява поддървото, което покрива всички възможни стойности на избрания атрибут.

Описаният подход към имплементация на процедурата НАУЧИ-ЕДНО-ПРАВИЛО осъществява търсене от общото-към-частното в пространството от възможни правила, с цел да намери едно правило с висока точност, макар и с непълно

покриване на данни. Както и при класификационните дървета има много различни начини за дефиниране на мярка за избор на "най-добрия" наследник. Например, можем, както и при ID3, да избираме най-добрия наследник на базата на ентропията, т.е. да избираме този наследник, който покрива множество от примери с най-малката ентропия.

Фиг. 5-1. Дървото на търсене

Разгледаният по-горе метод за търсене от общото-към-частното представлява евристичното търсене в дълбочина без възврат. Както при всеки евристичен метод за търсене това води до опасността за субоптималния избор, който се прави на всяка стъпка. За да намали този риск, можем да разширим нашият алгоритъм, за да извършва "лъчево" търсене, при което на всяка стъпка вместо един кандидат се поддържа списък от k най-добри кандидати. Т.е. на всяка стъпка от търсене наследниците (специализациите) се генерират за всеки от тези k най-добри кандидати, а полученото множество отново се намалява до k най-обещаващи кандидати. "Лъчевото" търсене пази най-обещаващите алтернативи на текущата най-добра хипотеза, така че всички техните наследници могат да бъдат разгледани на всяка стъпка от търсене. Този "лъчев" алгоритъм за търсене от общото-към-частното е използван в програмата CN2 (Clark & Niblet 1989). Алгоритмът е описан в Табл. 5-2.

НАУЧИ-ЕДНО-ПРАВИЛО(*Цел_атрибут, Атрибути, Примери, k*)

Връща едно правило, което покрива някои от *Примери*. Изпълнява "лъчево" търсене от общото-към-частното за намиране на най-доброто правило, водено от евристичната метрика ПОВЕДЕНИЕ.

- Инициализация:
  1. *Най-добрата_хипотеза* ← най-общата хипотеза *Ø*

2. *Кандидат_хипотези* ← *{Най-добрата_хипотеза}*
3. *Всички_ограничения* ← множество от всички ограничения *с* във вид *(а = v)*, където *а Є Атрибути*, *v* е значение на *а*, което се среща сред *Примери*

- **Докато** *Кандидат_хипотези* не е празно множество **направи**

1. Генерирай следващите по-специфични кандидат-хипотези
   - *Нови_кандидат_хипотези* ←
     За всяка *h* от *Кандидат_хипотези*
     За всяко *с* от *Всички_ограничения*
     - **Създай** специализацията на *h* чрез добавяне на ограничение *с*
   - **Изтрий** от *Нови_кандидат_хипотези* всички хипотези, които имат дубликати, са несъвместими или не са максимално специфични.
2. Обновяване на *Най-добрата_хипотеза*
   - За всяка *h* от *Нови_кандидат_хипотези* **направи**
     - **Ако** *h* е статистически значима върху *Примери*
       И (ПОВЕДЕНИЕ(*h, Примери, Цел_атрибут*) >
       ПОВЕДЕНИЕ(*Най-добрата_хипотеза*,*Примери, Цел_атрибут*))
       То *Най-добрата_хипотеза* ← *h*
3. Обновяване на *Кандидат_хипотези*
   - *Кандидат_хипотези* ← *k* най-добрите членове между *Нови_кандидат_хипотези* в съответствие с мярката ПОВЕДЕНИЕ

- **Върни** правило във вид:
  АКО *Най-добрата_хипотеза* ТО *Предсказание*,
  където *Предсказание* е най-често срещаната стойност на *Цел_атрибут* сред тези примери, които се покриват от *Най-добрата_хипотеза*

ПОВЕДЕНИЕ(*h, Примери, Цел_атрибут*)

- *h_примери* ← подмножество от *Примери*, покрити от *h*.
- **Върни** - *Entropy(h_примери)*, където ентропията се изчислява относно *Цел_атрибут*

Таблица 5-2. Алгоритъм за лъчево търсене на покриващото правило

Няколко забележки към алгоритъма за научаване на едно правило, описан в Табл. 5-2. Първо, в главния цикъл на алгоритъма всяка от разглежданите хипотези е конюнкцията от ограничения от типа *атрибут = стойност*. Всяка от тези конюктивни хипотези съответства на множество от кандидати – предусловия за правилото, подлежащо на научаване и оценявано чрез ентропията на примери, които то покрива. При търсенето се разглеждат все по-специфични хипотези, докато не се стигна до максимално специфичната хипотеза, която съдържа всички налични атрибути. Правилото, извеждано от алгоритъма, е правилото с най-голяма

стойност на мярката ПОВЕДЕНИЕ, което е било срещано в процеса на търсене – то съвсем *не е задължително* да бъде последната хипотеза, генерирана от алгоритъма. Пост-условието на изходното правило се избира само *на финалната стъпка* на алгоритъма, след като всички неговите предусловия (представени от променливата *Най-добрата_хипотеза*) са определени. Алгоритмът конструира пост-условие на правилото, за да предсказва тази стойност на целевия атрибут, която се среща най-често сред примерите, покрити от предусловията на правилото.

В оригиналния алгоритъм на CN2 кандидат-хипотезите се проверяват и за *статистическата значимост*, т.е. дали разпределението по класове на примерите, покрити от правилото не е случайно, а е резултат от съществуващата корелация между атрибутните стойности в тези примери и целевия атрибут (клас). По точно, се изчислява така наречена *статистика за относителната правдоподобност* (likelihood ratio statistic):

$$T(h,E) = 2\sum_{i=1}^{k} n_i \log_2(\frac{n_i}{e_i})$$

където:

n_i - броят на примери, принадлежащи към класа i, измежду всички примери, покрити от правилото h

e_i – *очакваният* брой на примери, принадлежащи към класа i, измежду всички примери, покрити от правилото h, при условие, че те са избрани по случаен начин, т.е. в съответствие с вероятностното разпределение на всички примери E, върху които се прилага процедурата за намиране на покриващото правило.

k – броят на възможни класове.

В математическата статистика е доказано, че ако общият броя на примери, покрити от правилото, е по-голям от 5, то тази статистика се подчинява на вероятностното хи-квадрат (χ^2) разпределение със *k-1* степени свобода. Правилото h се приема за статистически значимо, ако изчислената статистика е по-голяма и равна от табличната стойност на хи-квадрат разпределение със *k-1* степени свобода при зададената от потребителя степен на значимост α[1]:

$$T(h,E) \geq \chi^2_{\alpha,k-1}$$

И накрая, независимо от използване на "лъчевото" търсене, евристичното търсене все пак може да произвежда субоптимални правила. Обаче, дори и в този случай, алгоритмът ПОСЛЕДОВАТЕЛНО-ПОКРИВАНЕ е в състояние да научава колекцията от правила, които заедно покриват всички обучаващи примери, тъй като последователно извиква процедурата за научаване на едно правило на останали непокрити обучаващи примери.

5.1.2. Варианти

[1] Използването на математическата статистика при машинно самообучение е описано по-детайлно в лекциите 18 и 19.

Съществуват много различни варианти на описания по-горе базов алгоритъм за последователното покриване. Например, в някои случаи е желателно да имаме алгоритъм за научаване на правила, които покриват положителните примери и да назначават "по премълчаване" отрицателната класификация на всеки пример, който не се покрива от нито едно правило. Подобен подход е приложим, например, към задача за научаване на такова целево понятие, като "бременни жени, които могат да имат близнаци". В този случай пропорцията на положителните примери в общото множество от достъпни примери е сравнително малка, така че множеството от правила ще бъде по-компактно и по-разбираемо за хората, ако то идентифицира само класове на положителните примери, с класификацията по премълчаване за всички останали примери като отрицателните. Този подход отговаря на използваната в ПРОЛОГ стратегия "отрицание като пропадане", при която всеки израз, за който не може да бъде доказано, че приема стойност ИСТИНА, по премълчаване се подразбира, че приема стойност ЛЪЖА. За да бъдат научени правила, предсказващи само една единствена стойност на целевия атрибут, нашия алгоритъм НАУЧИ-ЕДНО-ПРАВИЛО трябва да бъде модифициран по такъв начин, че да може да приема като вход желаната стойност на целевия атрибут. "Лъчевото" търсене от общото-към-частното ще се осъществява по същия начин, като трябва да бъде променена само метриката ПОВЕДЕНИЕ, оценяваща текущите хипотези. Обърнете внимание, че за новата задача използването на отрицателната ентропия е вече неподходящо, тъй като тази метрика ще назначава най-много точки на хипотези, които покриват само отрицателните примери или само положителните. По тази причина, за оценяването на хипотези по-подходящо да бъде използвана мярката, която измерва пропорцията на положителните примери, покривани от хипотези.

Другият вариант на подхода за научаване на правила чрез последователно покриване е имплементиран във фамилията от алгоритми, наречени AQ (Michalski 1969, 1986), които са предшествениците на разгледания по-горе алгоритъм CN2. Подобно на CN2, AQ научава дизюнктивното множество от правила, които заедно покриват целевата функция. Обаче AQ се отличава от описания по-горе алгоритъм по няколко важни характеристики.

Първо, в отличието от алгоритъма ПОСЛЕДОВАТЕЛНО-ПОКРИВАНЕ, AQ явно търси правила, покриващи *конкретно значение на целевия атрибут*, научавайки последователно по едно дизюнктивно множество от правила за всяка стойност на целевия атрибут.

Второ, начинът за научаване на едно правило, използван от AQ, е различен от процедурата НАУЧИ-ЕДНО-ПРАВИЛО. Докато извършва "лъчевото" търсене от общото-към-частното за всяко правило, AQ използва *един единствен пример* за фокусиране на това търсене. В частност, при търсене на все по-специфични хипотези се разглеждат само тези атрибути, които удовлетворяват *избрания положителен пример*. Всеки път при научаване на новото правило се разглежда някакъв нов положителен пример, който остава още не покрит от научените до сега

правила. Този пример се ползва като ядро, управляващо търсене на новото правило, покриващо още непокрития дизюнкт.

5.2. Сравнение на подходи за научаване на множество от правила

Методите за последователното покриване и използване на класификационните дървета предлагат множество от варианти за научаване на правила. В този раздел ще разгледаме няколко основни размерности, описващи пространството от възможни подходи към тази задача.

Първо, алгоритмите за *последователното покриван*е научават по едно правило на цикъл, премахвайки покритите от него примери и повтаряйки процеса върху останалите примери. Алгоритмите за научаване на класификационните дървета научават пълното множество от дизюнкти едновременно като част от един единствен процес на търсене за някое приемливо класификационно дърво. По тази причина, алгоритмите от този тип (например ID3) могат да се нарекат *алгоритмите за едновременно покриване*, за разликата от алгоритмите за последователното покриване от типа на CN2. Кой тип алгоритмите е за предпочитане? Ключовата разлика е в избора, който се прави на най-примитивната стъпка от търсене. Така ID3 избира на всяка стъпка между *алтернативни атрибути*, сравнявайки *начина на разделяне на данни*, които те правят. От своя страна CN2 избира между *алтернативни двойки атрибут-стойност*, сравнявайки *подмножествата от данни*, *които те покриват*. Един от начини да разбере важността на разликата между двата подхода, е да се сравни броя на различни избори, които се правят от двата алгоритъма, за да научи едно и също множество от правила. Така за да научи множество от n правила, всяко от които съдържа k теста на атрибутните стойности в своите предусловия, алгоритмите за последователното покриване ще направят $n*k$ примитивни стъпки на търсене, приемайки по едно независимо решение за избор на всяко предусловие за всяко правило. От своята страна, алгоритмите за едновременното покриване ще направят много по-малко независими избори, тъй като избор на един решаващ възел в дървото съответства на избора на предусловието за цялото множество от правила, асоциирани с този възел. С други думи, ако решаващият възел на класификационното дърво проверява атрибут с m възможни стойности, изборът на този възел съответства на избора на едно предусловие за всяко от m съответни правила. Следователно, алгоритмите за последователното покриване извършват по-голям брой независими избори отколко алгоритмите за едновременното покриване. Тогава остава въпрос, кой тип алгоритми е за предпочитане? Отговорът може да зависи от обема на наличните обучаващи данни. Ако обемът е голям, той може да поддържа по-големия брой на независими избори, необходими за работата на алгоритми за последователното покриване, докато ако данните са малко, "съвместното" приемане на решения за предусловията на различни правила може да се окаже по-ефективно. Като един допълнителен аргумент за избора може да бъде използван зависещият от конкретната задача въпрос, дали е желателно да имаш правила,

които проверяват едни и същи атрибути. При алгоритмите за едновременното покриване (от типа на класификационните дървета) такива правила се пораждат. При алгоритмите за последователно покриване – не.

Втората размерност за сравнение на различни подходи към научаване на правила е направлението, в което се осъществява търсене от процедурата НАУЧИ-ЕДНО-ПРАВИЛО. В разгледаните алгоритми то е *от-общото-към-частното*. Обаче в другите алгоритми (например FIND-S), то е *от-частното-към-общото*. Предимството на подхода от общото към частното е в съществуване на само една най-обща хипотеза, от която започва търсене, до като при обратния подход брой на максимално специфични хипотези е равен на броя на обучаващите примери. При това положение е неясно, от коя най-специфична хипотеза трябва да бъде започнато търсене. Една от програми, използващи търсене от частното към общото – GOLEM (Muggleton & Feng 1990) решава този въпрос чрез случаен избор на няколко положителни примера за инициализиране и направляване на търсене. Най-добрата хипотеза се избира от получените при тези неколкократни случайни избори.

Третата размерност е дали процедурата НАУЧИ-ЕДНО-ПРАВИЛО извършва търсене в пространство от синтактически коректни хипотези от типа *породи-и-провери* (както се прави в разгледаната по-горе имплементацията) или търсенето е *водено-от-примери*, при което отделният обучаващ пример ограничава пораждането на възможни хипотези. Примери за втория подход са алгоритмите FIND-S, елиминиране на кандидати и AQ. Във всеки от тези алгоритми пораждането или ревизията на хипотези се управлява от анализа на отделния обучаващ пример и, като резултат, получената хипотеза коректно класифицира този единствен пример. Това контрастира с търсенето, реализирано в процедурата НАУЧИ-ЕДНО-ПРАВИЛО, при която хипотезите-наследници се пораждат само на базата на синтаксиса, избран за представяне на хипотези. Обучаващите данни се използват само *след* пораждането на кандидат-хипотези и се прилагат за избор между кандидатите на базата на тяхното поведение върху цялото множество от обучаващите примери. Едно важно предимство на този подход е, че всеки избор при търсенето се прави на базата на поведението на хипотези върху *множеството* от примери, намалявайки по този начин влияние на *зашумени* примери. От своя страна, алгоритмите, при които търсене се управлява от примери, значително по-лесно могат да бъдат подведени от един единствен зашумен пример и, следователно, по-малко устойчиви към наличието на грешки в обучаващите данни.

Четвъртата размерност е дали и как трябва да се прави допълнително подрязване на правила. Както и в случая с класификационните дървета е възможно процедурата НАУЧИ-ЕДНО-ПРАВИЛО да научава правила, които имат много добро поведение върху обучаващите примери, но значително по-лошо – върху нови, неизвестни при обучение примери. Едно възможно решение на този проблем е прилагане на допълнително подрязване на всяко правило след тяхното научаване. В частност, някои от предусловия на правилото могат да бъдат премахнати, ако това води до

подобряване на поведението върху отделното, независимото от обучаващото множество, множество от примери.

И последната размерност е дефиницията на конкретната мярка ПОВЕДЕНИЕ на правила, избрана за управление на търсене в процедурата НАУЧИ-ЕДНО-ПРАВИЛО. Някои от най-често срещани функции са: относителната честота, *m*-приближение на точността и ентропията. В CN2 се използва мярка за отрицателната ентропия, комбинирана с тест за статистическата значимост, която „наказва" стойностите, изведени от недостатъчното количество данни.

5.3. Научаване на правила, описани чрез логиката от първия ред

До сега ние разглеждахме алгоритми за научаване на множества от *пропозиционални* правила, т.е. правила, описани на езика на атрибутивната логика. Предусловията на правилата са конюнкции от ограничения атрибут –стойност и не могат да съдържат променливи. В този раздел ще се запознаем с алгоритми за научаване на правила, представени на езика на логиката от първия ред, който е значително по-изразителен от езика на атрибутивната логика. Индуктивното научаване на правилата, описани чрез логиката от първия ред, често наричат *индуктивно логическо програмиране* (ИЛП), тъй като този процес може да се разглежда като автоматичен извод на ПРОЛОГ-програми от примери.

5.3.1. Терминология

За да видим предимството от представяне на правила с езика на логиката от първия ред пред пропозиционалното (без променливи) представяне, да разгледаме задачата за научаване на целевото понятие *дъщеря(X, Y)*, определено върху двойки хора *X* и *Y*. Стойността на *дъщеря(X, Y)* е *истина*, когато *Y* е дъщеря на *X* и *лъжа* в останалите случаи. Нека всеки човек е описан с данни, представени чрез атрибути *име, майка, баща, мъж, жена*. Следователно, всеки обучаващ пример ще състои от описанието на двойки хора в термините на тези атрибути, както и със стойността на целевия атрибут *дъщеря*. Например, един положителен пример, че Катерина е дъщеря на Николай, ще изглежда по следния начин (ще използваме синтаксис близък до ПРОЛОГ):

$<име_1 = Катерина, майка_1 = Виолета, баща_1 = Николай, мъж_1 = лъжа,$
$жена_1 = истина, име_2 = Николай, майка_2 = Ценка, баща_2 = Иван,$
$мъж_2 = истина, жена_2 = лъжа, дъщеря_{2,1} = истина>$

Индексите на атрибути показват, кои от хората са описани.

Ако съставим обучаващото множество от подобни примери на целевото понятие $дъщеря_{2,1}$ и го предоставим на някой алгоритъм за научаване на пропозиционални

правила от типа на CN2 или C4.5, то като резултат ще получим множеството от много специфични правила от вида на :

$$АКО\ \ (баща_1 = Николай) \wedge (име_2 = Николай) \wedge (жена_1 = истина)$$

$$ТО\ \ \ дъщеря_{2,1} = истина$$

Макар, че всички тези правила са правилни, те са много специфични и на практика не могат да се използват за класификацията на двойки хора, отлични от описаните в обучаващото множество. Проблемът е в това, че в пропозиционалната логика липсват средства за описание на общи *релации* между стойностите на атрибути. Напротив, програмата, използваща представяне чрез логиката от първия ред, ще може да научи следното общо правило:

$$АКО\ \ баща(Y, X) \wedge жена(Y)\ \ \ ТО\ \ \ дъщеря(X, Y)$$

където X и Y са променливи, които могат да се свържат с произволен човек.

Преди да разгледаме конкретни алгоритми за научаване на правила, нека да въведем базовата терминология от областта на формалната логика:

- Всеки **правилно сформиран израз** се състои от *константи* (напр. *'Иван', 23, кресло*), *променливи* (напр. *X*), *предикати* (напр. *жена('Пенка')*) и *функции* (напр. *възраст* - като *възраст('Пенка')*)
- **Терм** е произволна константа, променлива или функция, приложена към произволен терм. Например, *'Николай', X, възраст('Николай'), възраст(X).*
- **Литерал** е произволен предикат (или неговото отрицание), приложен към произволно множество от терми. Например, *жена('Пенка'),* $\neg$*жена(X), по-възрастна_от(възраст('Пенка'), 20)*.
- **Основен литерал** е литерал, който не съдържа никакви променливи (напр., $\neg$*жена('Пенка')*).
- **Отрицателен литерал** е литерал, съдържащ отрицанието на предикат (напр., $\neg$*жена('Пенка')*), $\neg$*жена(X)*).
- **Положителен литерал** е литерал без знака за отрицание.
- **Клауза** е произволна дизюнкция от литерали $M_1 \vee ...M_n$, чиито променливи са универсално квантифицирани.
- **Клауза на Хорн** е клауза, съдържаща най-много един положителен литерал:
  $H \vee \neg L_1 \vee \neg L_2 \vee ...\neg L_n$,
  където H е положителния литерал, а $\neg L_1, \neg L_2, ..., \neg L_n$ са отрицателните литерали. H се нарича *глава* или *консеквент* на клаузата на Хорн, а конюнкцията от литерали $L_1 \wedge ... \wedge L_n$ - *тяло* или *антицедент* на клаузата.
- За всеки литерал A и B, изразът $(A \leftarrow B)$ е еквивалентен на $(A \vee \neg B)$, а изразът $\neg(A \wedge B)$ е еквивалентен на $(\neg A \vee \neg B)$. По тази причина, една клауза на Хорн може да се запише във вид: $H \leftarrow (L_1 \wedge ... \wedge L_n)$, което е еквивалентно на използваната по-рано нотация за правила: АКО $L_1 \wedge ... \wedge L_n$ ТО H .
- **Субституция** е функцията, която замества променливи с терми. Например, субституцията *{X/3, Y/Z}* замества променливата X с терм 3, а променливата Y – с терм Z. При зададени субституция θ и литерал L ще

използваме символ $L\theta$ за означаване на резултат от прилагане на субституцията θ към L.

- **Унифицираща субституция** за два литерала L_1 и L_2 е произволна субституция θ, такава че $L_1\theta = L_2\theta$.

5.4. Научаване на правила от първия ред: програма FOIL

Съществува множество алгоритми за научаване на правила, представени чрез логиката от първия ред. Ние ще разгледаме един от тях – FOIL (Quinlan 1990), който се базира на подход, много близък до алгоритмите ПОСЛЕДОВАТЕЛНО-ПОКРИВАНЕ и НАУЧИ-ЕДНО-ПРАВИЛО, разгледани по-рано. Фактически FOIL е едно естествено разширение на тези алгоритми към представяне чрез логиката от първия ред. Формално хипотезите, които научава FOIL, са множества от правила от първия ред, като всяко правило е подобно на клаузата на Хорн със следните два изключения. Първо, правилата, научавани от FOIL, са по-ограничени от общите клаузи на Хорн, тъй като на литералите в тях не е позволено да съдържат функционални символи (това намалява сложността на претърсваното пространството от хипотези). Второ, правилата на FOIL са по-изразителни от клаузите на Хорн, тъй като допускат наличието в тялото на правилото на отрицателните литерали. FOIL е бил прилаган в различни предметни области, например за научаване как да се различават коректни от некоректни шахматни позиции.

Алгоритмът на FOIL е представен в Таблица 5–3. Обърнете внимание, че външният цикъл съответства на вариант на алгоритъма за последователното покриване, описан по-рано, т.е. той научава нови правила едно след друго, изтривайки положителните примери, покрити от наученото правило, преди да опита да научи следващото правило. Вътрешният цикъл съответства на вариант на процедурата НАУЧИ-ЕДНО-ПРАВИЛО, разширен по начина, позволяващ да научава правила от първия ред. Обърнете внимание на няколко незначителни разлики между FOIL и дискутираните по-рано алгоритми. В частност, FOIL търси само правила, предсказващи, кога стойността на целевия литерал ще бъде *истина*, докато по-ранният алгоритъм търсеше както правила, предсказващи, кога литералът е *истина*, така и когато той е *лъжа*. Освен това, FOIL реализира градиентния метод на търсене, а не "лъчевото" търсене, което е еквивалентно на "лъчевото" търсене с ширината на лъча равна 1.

FOIL(Цел_атрибут, Предикати, Примери)

- *Пол* ← тези от *Примери*, за които *Цел_атрибут* е *истина*
- *Отр* ← тези от *Примери*, за които *Цел_атрибут* е *лъжа*
- *Научени_правила* ← *{}*
- **Докато** *Пол* **направи**

  Научи Ново_Правило

- *Ново_Правило* ← правилото, което предсказва Цел_атрибут без всякакви предусловия
- *Ново_Правило_Отр* ← *Отр*
- **Докато** *Ново_Правило_Отр* **направи**
  Добави нов литерал за да специализира Ново_Правило
  - *Кандидат_Литерали* ← генерирай кандидат-литерали за *Ново_Правило* на базата на *Предикати*
  - $Най-добрия\_Литерал \leftarrow \arg\max_{L \in Кандидат\_Литерали} FOIL\_Gain(L, Ново\_Правило)$
  - добави *Най – добрия _ Литерал* к предусловията на *Ново_Правило*
  - *Ново_Правило_Отр* ← подмножество от *Ново_Правило_Отр*, което удовлетворява предусловията на *Ново_Правило*
- *Научени_правила* ← *Научени_правила* + *Ново_Правило*
- *Пол* ← *Пол* – {членове на *Пол*, покрити от *Ново_Правило*}
- **Върни** *Научени_правила*

Таблица 5-3. Алгоритмът на FOIL

Търсенето на FOIL в пространството на хипотези най-добре се разбира от йерархичната гледна точка. При всяка итерация външният цикъл добавя по едно ново правило към съществуващата дизюнктивна хипотеза – *Научени_Правила*. Ефектът от добавянето на правило се състои в обобщаване на текущата дизюнктивна хипотеза (т.е. се увеличава броя на примери, които тя класифицира като положителни) чрез добавяне на новия дизюнкт. От този гледна точка търсенето е *от-частното-към-общото* в пространството на хипотези, започвайки с най-специфичната празна дизюнкция и завършвайки, когато хипотезата става достатъчно обща, за да покрива всички положителни обучаващи примери. Вътрешния цикъл на FOIL изпълнява по-фино търсене, за да определи точната дефиниция на всяко ново правило. Този вътрешен цикъл осъществява търсене във второто пространство от хипотези, състоящо от конюнкции на литерали, за да намери конюнкцията, която ще формира предусловие на новото правило. В това пространство на хипотези FOIL осъществява градиентно търсене от *общото-към-частното*, започвайки с най-общо възможно предусловие (празно предусловие), добавяйки последователно към него по един литерал за итерацията, за да специализира правилото, докато то не започва да избягва (да не покрива) всички отрицателни примери.

Двете най-съществени разлики между FOIL и разгледаните по-рано алгоритми ПОСЛЕДОВАТЕЛНО-ПОКРИВАНЕ и НАУЧИ-ЕДНО-ПРАВИЛО произтичат от изисквания за нагаждане към представяне на правила чрез логиката от първия ред. Те са:

1. При търсенето от-общото-към-частното на новото правило FOIL изпълнява и други стъпки, необходими за да формира кандидат-специализации на правилото. Тази разлика произтича от необходимостта да се нагоди към използването на променливи в предусловията на правила.
2. FOIL използва другата мярка за ПОВЕДЕНИЕ – FOIL_Gain, която се отличава от мярката за ентропия, използвана в НАУЧИ-ЕДНО-ПРАВИЛО. Тази разлика е следствие от необходимостта да различава между различните свързвания на променливите в правилото и от факта, че FOIL търси правила, които покриват само положителните примери.

А сега да разгледаме тези разлики по-детайлно.

5.4.1. Пораждане на кандидат-специализации в FOIL

За формиране на кандидат-специализации на текущото правило FOIL поражда множество нови литерали, всеки от които може да бъде по-отделно добавен към предусловия на правилото. Да предположим, че текущото правило е във вид: $p(X_1, X_2, ..., X_k) \leftarrow L_1, ..., L_n$. FOIL поражда кандидат-специализации на това правило чрез разглеждане на нови литерали L_{n+1}, които са представени в една от следните форми:

- $q(V_1, ..., V_r)$, където q е име на някой предикат, срещан в *Предикати*, а V_i са или нови променливи, или променливи, които вече присъстват в правилото. Най-малко една от V_i в създавания литерал трябва вече да е съществувала като променливата в правилото.
- *равни*(X_j, X_k) – литерал, свързващ вече съществуващите в правилото променливи X_j и X_k
- Отрицанието на всеки от предишните два вида литерали.

Като илюстрация да разгледаме научаване на правила за предсказване на целевия литерал *внучка(X,Y)*, където другите предикати, използвани за описание на примери, са *баща* и *жена*. Търсенето от общото-към-частното в FOIL започва с най-общото правило:

внучка(X,Y) ←

което утвърждава, че *внучка(X,Y)* е истина за всеки X и Y. За да специализира това правило, описаната по-горе процедура генерира следните литерали като кандидати за добавяне към неговото предусловие: *равни(X, Y), жена(X), жена(Y), баща (X, Y), баща(Y, X), баща(X, Z), баща(Z, X), баща(Y, Z), баща(Z,Y)*, както и отрицанията на всички тези литерали (напр. ¬*равни(X,Y)*). Z е новата променлива, докато X и Y вече съществуват в правилото.

Да предположим, че в резултат на евристичната оценка FOIL избира сред тези литерали *баща(Y,Z)* като най-обещаващ кандидат за специализацията на правилото:

внучка(X, Y) ← *баща(Y, Z)*

При генериране на новите кандидат-литерали за по-нататъшната специализация на това правило FOIL ще разглежда всички литерали, разгледаните на предишната

стъпка, плюс следните нови литерали: *жена(Z), равни(Z, X), равни(Z, Y), баща(Z, W), баща(W, Z)* както и техните отрицания. Те се разглеждат защото Z е вече съществуващата променлива в правилото и, следователно, FOIL въвежда новата променлива W.

Ако на тази стъпка FOIL избира литерал *баща(Z, X)*, а на следващата – литерал *жена(Y)*, то това ще доведе до следното правило, покриващо само положителните примери и, следователно, водещо до прекратяване на търсене за по-нататъшните специализации:

$$внучка(X,Y) \leftarrow баща(Y,Z) \wedge баща(Z,X) \wedge жена(Y)$$

В този момент FOIL изтрива всички положителни примери, покрити от новото правило. Ако остават още непокрити положителни примери, алгоритмът започва новия цикъл от своето търсене от общото-към-частното за научаване на новото правило.

5.4.2. Управление на търсене в FOIL

За избор на най-обещаващ литерал от множеството на кандидат-литерали, пораждани на всяка стъпка, FOIL разглежда поведението на правилото върху обучаващите данни. За да направи това, той разглежда всички възможни свързвания на всяка променлива в правилото. За да илюстрираме този процес, да разгледаме отново примера за търсене на множество от правила за целевия литерал *внучка(X,Y)*. Да предположим, че обучаващите данни съдържат следното множество от прости утвърждения (предикат *p(X, Y)* означава, че "*p* на *X* е *Y*").

внучка('Виктор', 'Катерина') баща('Катерина','Николай')
баща('Иван', 'Николай') жена('Катерина') баща('Николай','Виктор')

Да направим също така допускане за "затворен свят", утвърждаващо, че всеки литерал, съдържащ предикат *внучка, баща* или *жена* и константи *'Виктор', 'Катерина'* и *'Николай'*, който не е указан по-горе, се предполага да бъде в състояние "*лъжа*" (т.е. ние неявно добавяме утвърждения $\neg$*внучка('Николай', 'Виктор'),* $\neg$*внучка('Николай', 'Николай')* и т.н.).

За избор на най-добрата специализация на текущото правило FOIL разглежда всеки начин, различаващ от този, по който променливите в правилото могат да бъдат свързани с константи, фигуриращи в обучаващите примери. Например, в началната стъпка, когато правило има вид

внучка(X,Y) $\leftarrow$

променливите в правило X и Y не са ограничени с никакви предусловия и, по тази причина, могат да се свържат във всяка комбинация с четири константи *'Виктор', 'Николай', 'Катерина'* и *'Иван'*. При наличието на тези четири константи са възможни 16 варианта на свързвания на променливи в това правило. Свързване *{X/'Виктор', Y/'Катерина'}* отговаря на свързването в един положителен пример, тъй като обучаващите данни съдържат утвърждението *внучка('Виктор',*

'Катерина'). Други 15 свързвания, позволени от това правило (например *{X/'Иван', Y/'Николай'}*) водят до пропадане на правилото (отрицателното свидетелство) тъй като сред обучаващите данни не се намира съответното утвърждение (*внучка('Иван', 'Николай'*)).

На всеки етап правилото се оценява на базата на тези множества от положителни и отрицателни свързвания на променливи, като предимството се дава на правила, които имат повече положителни и по-малко отрицателни свързвания. След добавяне на новия литерал към правилото, множеството на свързвания се променя. Обърнете внимание, че когато се добавя литерал, въвеждащ нова променлива, свързванията на такова правило се увеличават по своята дължина (например, ако към горното правило се добави *баща (Y, Z)*, то началното свързване *{X/'Виктор', Y/'Катерина'}* ще стане по-дълго: *{X/'Виктор', Y/'Катерина', Z/'Николай'})*. Освен това, новата променлива може да се свърже с различни константи, така че броят на свързвания, асоциирани с новото правило, може да бъде по-голям от броя на свързвания, асоциирани с оригиналното правило.

Оценъчната функция, използвана в FOIL за преценяване на полезността от добавяне на новия литерал, се базира на броя на положителни и отрицателни свързвания, покрити преди и след добавяне на новия литерал. Да разгледаме някое правило R и кандидат-литерал L, който може да бъде добавен към тялото на R. Нека R' е новото правило, създадено чрез добавянето на този литерал към R. Стойността *FOIL_Gain(L, R)* на добавянето на L към R се дефинира като:

$$FOIL\_Gain(L,R) \equiv t\left(\log_2 \frac{p_1}{p_1+n_1} - \log_2 \frac{p_0}{p_0+n_0}\right)$$

където p_0 е броят на положителни свързвания на правило R, n_0 – брой на отрицателни свързвания на R, p_1 и n_1 са, съответно, броят на положителни и отрицателни свързвания на правилото R'. t е броят на положителните свързвания на правилото R, които остават покрити след добавяне на литерала L към R. При въвеждане на новата променлива в R чрез добавяне на L, всички първоначални свързвания се разглеждат като покрити до тогава, докато някои от нови свързвания, разширяващи ги, присъстват в свързванията на R'.

Функцията *FOIL_Gain* има директна интерпретация в термините на теорията на информация. Съгласно тази теория, $-\log_2 \frac{p_0}{p_0+n_0}$ е минималното количество на битове, необходимо за кодиране на класификацията на едно произволно положително свързване измежду свързванията, покрити от правилото R. По същия начин, $-\log_2 \frac{p_1}{p_1+n_1}$ е количеството на битове, необходими за кодирането, ако свързването е измежду тези, покрити от правилото R'. Тъй като t е точно броя на положителни свързвания, покрити от R, които остават покрити и от R', то *FOIL_Gain* може да се разглежда като печалба (намаление) в общия брой на

битове, необходими за кодиране на класификацията на всички положителни свързвания в R, получено в резултат от добавянето на L към R.

5.4.3. Научаване на рекурсивни правила

В предишния раздел не са разглеждани случаи, когато новите литерали, добавяни към тялото, могат да сочат отново към целевия предикат (т.е. предикат, участващ в главата на правилото). Обаче, ако включим целевия предикат във входния списък *Предикати*, то FOIL ще го разглежда при генериране на кандидат-литерали. Това ще позволи да бъдат формирани рекурсивни правила – т.е. правила, използващи същите предикати в тялото и главата на правилото. Например, рекурсивната дефиниция на понятие *наследник* може да бъде получена от следното множество от правила:

АКО *баща(X,Y)* ТО *наследник (X,Y)*
АКО *баща* $(X, Z) \wedge$ *наследник*(Z, Y) ТО *наследник(X,Y)*

При зададени подходящи обучаващи примери, тези два правила могат да бъдат научени по начина, сходен с този, описан за понятие *внучка*. Обърнете внимание, че второто правило е потенциално достъпно на FOIL при търсене, ако предикат *наследник* е включен в списъка *Предикати,* който определя, кои предикати могат да бъдат разглеждани при генериране на новите литерали. Дали това конкретно правило ще бъде научено или не, зависи от това, дали тези конкретни литерали ще бъдат по-добри, от гледната точка на оценъчната функция, от другите кандидати, разглеждани от FOIL в процеса на евристичното търсене на по-специфични правила. Cameron-Jones и Quinlan (1993) разглеждат някои примери, в които FOIL успешно научил множества от рекурсивни правила, а също така обсъждат и въпроси, как да бъде избегнато научаване на правила, водещи до безкрайната рекурсия.