Лекция 9. Предсказване чрез регресионни дървета

9.1. Какво е предсказването?

Едни от най-често срещани ИЗД задачи са класификация и предсказване. *Класификацията* предполага наличието на множеството от обекти, описвани чрез определени атрибути и принадлежащи към различни класове. Целевият атрибут (клас) приема дискретни (символни) стойности и е известен за всеки обект. Целта е да бъдат построени класификационни модели (класификатори), които назначават коректен клас на по-рано неизвестни обекти - обекти с неизвестна стойност на целевия атрибут. Класификационните модели предимно се използват за предсказващото моделиране. Стойността на целевия атрибут може да бъде зададена предварително (например от потребителя) или изведена от сегментацията.

Класификационните модели могат да се представят във вид на класификационни дървета, правила, невронни мрежи и т.н. Начини на построявана (научаване) и използване на такива класификационни методи бяха подробно разгледани в курса по машинно самообучение.

Предсказването е много сходно с класификацията, като единствената разлика е в това, че при предсказването целевият атрибут (клас) приема не дискретни, а *непрекъснати* стойности. Това означава, че целта на предсказването е да се намери числовата стойност на целевия атрибут за по-рано неизвестни обекти. Този тип задачи често се нарича *регресия*.

В общия случай, задачата за регресия (*регресионен анализ*) може да бъде дефинирана като *задача за научаване на зависимости, свързващи някоя целева променлива (зависима променлива, отговор) с множество от независими (предсказващи, входни) променливи*. Целта на регресионен анализ е създаване на модела на тези зависимости. Когато входната променлива е само една, задачата се нарича *проста регресия*, когато ги няколко – *множествена (multivariate) регресия*. Регресионният анализ е една доста стара задача, която води своето начало от работите на британския антрополог и метеоролог сър Франсис Галтон (1822-1911). В тази лекция ще разгледаме най-често използвани методи за научаване на модели на проста и множествена регресия, използвани за предсказване на стойността на целевата променлива, като основното внимание ще бъде върху регресионни дървета.

9.2. Формализация на регресионната задача

Задача на един регресионен метод е да създаде определен модел, използвайки зададената извадка от обекти (обучаващо множество), принадлежащи към някоя неизвестна регресионна функция. Обучаващото множество се състои от двойки във вид $<\mathbf{x}_i, y_i>$, i = 1, …, n, където $\mathbf{x}_i$ е p-мерен вектор от стойностите на атрибути (предсказващи променливи), а y_i е съответната стойност на целевият

атрибут (зависима променлива). При регресионен анализ често се използва матрична нотация, при която чрез **X** се означава матрицата с размерност *n x p*, чийто i-я ред е входния вектор. Стойностите на целевия атрибут от обучаващите примери се представят като n x 1 матрица **Y**. Регресионният модел е функция, изобразяваща един входен вектор $\mathbf{x}_i \in \Re^p$ в някое реално число $y_i \in \Re^1$.

Регресионната зависимост между атрибутите и целевата променлива обикновено се описва като:

$y_i = r(\boldsymbol{\beta}, \mathbf{x}_i) + \varepsilon_i$ (9.1)

където $r(\boldsymbol{\beta}, \mathbf{x})$ е някой регресионен модел върху входните променливи $\{\mathbf{X}_i\}_{i=1}^p$ с параметри $\boldsymbol{\beta}$, а ε_i – грешки в наблюдения

Основната цел на един регресионен метод е да бъдат намерени "най-добрите" параметри на модела $\boldsymbol{\beta}$ в съответствие с избрания критерий за качеството. Макар, че това пристрастие при търсене може да включва както оценъчната грешка при предсказването (т.е. поведението на модела върху тестови примери), така и простота на модела, повечето от съществуващи регресионни методи търсят най-добрите параметри само на база на предсказващата сила на модела.

9.2.1. Мерки за оценяване на точността на регресионни модели

Тъй като целевата променлива при регресионната задача приема числени стойности, то всички най-често използвани мерки за оценяване на точността на регресионни модели се базират върху разликите между предсказвана и истинската стойност на тази променлива.

Средно абсолютно отклонение (MAD – Mean Absolute Deviation) е мярката за грешка, която оценява модела чрез осредняването на абсолютни отклонения на истинските стойности на целевата променлива от техните предсказани стойности:

$$MAD(r) = \frac{1}{n}\sum_{i=1}^{n} | y_i - r(\boldsymbol{\beta}, \mathbf{x}_i) | \quad (9.2)$$

Класическата мярка, използвана в регресионния анализ, е *средна квадратична грешка* (MSE – Mean Squared Error):

$$MSE(r) = \frac{1}{n}\sum_{i=1}^{n} (y_i - r(\boldsymbol{\beta}, \mathbf{x}_i))^2 \quad (9.3)$$

Методите, използващи тази мярка като критерий за минимизация при търсене на параметри на регресионни модели, традиционно се наричат *регресионни методи на най-малките квадрати* (Least Squares Regression).

Друга, често използвана мярка за грешка е о*тносителна средна квадратична грешка* (RMSE - Relative Mean Squared Error):

$$RMSE(r) = \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - r(\boldsymbol{\beta}, \mathbf{x}_i))^2}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2} = \frac{MSE(r)}{MSE(\bar{y})}, \quad (9.4)$$

където $\bar{y}$ е средната стойност на **y** в обучаващото множество.

Тази мярка дава относителната стойност на грешката при "напасване" чрез модела – стойността между нула и единица показва, че *r* върши по-добра работа от колко да предсказва само средната стойност на **y**.

9.3. Съществуващи регресионни методи

9.3.1. Статистически методи

Както вече беше казано, регресията е една от основните задачи, изучавана при статистическия анализ на данни. По тази причина съществува огромно количество работи в тази област, както и множество от различни подходи. Най-често използваните от тях са глобални параметрични подходи, не-параметрични подходи и адитивни модели.

9.3.1.1. Глобални параметрични подходи

Глобалните параметрични подходи опитват да намерят един единствен функционален модел, който добре "пасва" към всички зададени обучаващи данни. Те се базират върху строги предположения за вида на неизвестна регресионна (целева) функция, което може да доведе до низка предсказваща точност на модела, когато тези предположения не са изпълнени. Обаче, подобните подходи дават сравнително прости функционални модели, които са лесни за интерпретиране и позволяват намирането на лесни от изчислителната гледна точка решения.

Класическия пример за такъв глобален подход е широко известен *линеен регресионен модел*, който предполага, че целевата функция е някоя линейна комбинация от своите входните променливи – атрибути.

Проста линейна регресия

Простата линейна регресия предполага, че регресионната (целева) функция може да се моделира като линейната функция само от една единствена променлива, т.е.

$$r(\boldsymbol{\beta}, x) = \beta_1 x + \beta_0 \quad (9.5)$$

В качеството на критерий за избор на параметрите се използва критерия за най-малката квадратична грешка (LSE):

$$LSE = \sum_{i=1}^{n}(y_i - r(\boldsymbol{\beta}, x_i))^2 = \sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0))^2 \quad (9.6)$$

Решението на тази задача може да се търси чрез намирането на частни производни на критерия по β_1 и β_0, установяването ги в нула и решаването на съответната система от линейни уравнения:

$$\begin{aligned}\frac{\partial LSE}{\partial \beta_0} &= -2\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0) = 0 \\ \frac{\partial LSE}{\partial \beta_1} &= -2\sum_{i=1}^{n}(y_i - \beta_1 x_i - \beta_0)x_i = 0\end{aligned} \quad (9.7)$$

Тези уравнения могат да бъдат преписани в матричната форма, наричана *нормално уравнение*:

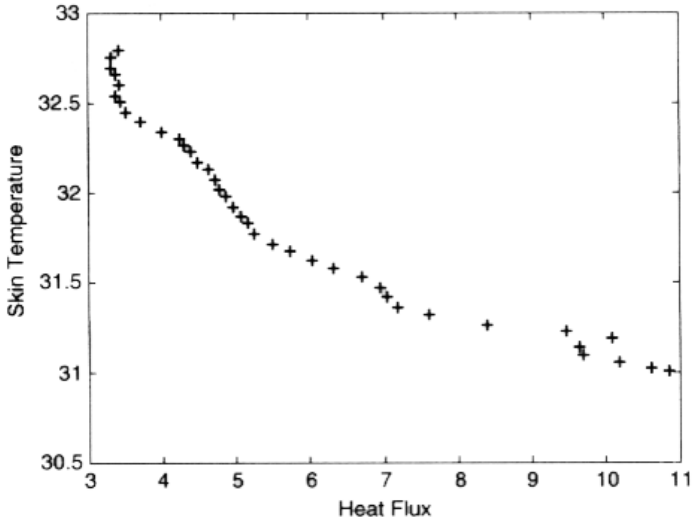
$$\begin{pmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{pmatrix}\begin{pmatrix} \beta_1 \\ \beta_0 \end{pmatrix} = \begin{pmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{pmatrix} \quad (9.8)$$

За да стане изложението по-ясно, да разгледаме един конкретен пример от областта на физиологията. На фиг. 9-1 са представени данни и съответната графика на данни от измервания на топлинен поток и температурата на кожата на един пациент по време на сън.

Heat Flux	Skin Temperature
10.858	31.002
10.617	31.021
10.183	31.058
9.7003	31.095
9.652	31.133
10.086	31.188
9.459	31.226
8.3972	31.263
7.6251	31.319
7.1907	31.356
7.046	31.412
6.9494	31.468
6.7081	31.524

Heat Flux	Skin Temperature
6.3221	31.581
6.0325	31.618
5.7429	31.674
5.5016	31.712
5.2603	31.768
5.1638	31.825
5.0673	31.862
4.9708	31.919
4.8743	31.975
4.7777	32.013
4.7295	32.07
4.633	32.126
4.4882	32.164

Heat Flux	Skin Temperature
4.3917	32.221
4.2951	32.259
4.2469	32.296
4.0056	32.334
3.716	32.391
3.523	32.448
3.4265	32.505
3.3782	32.543
3.4265	32.6
3.3782	32.657
3.3299	32.696
3.3299	32.753
3.4265	32.791



Фиг. 9.1 Данните от измервания на топлинния поток и температура на кожата на пациента

Задачата е да се предскаже температурата на кожата на базата на измервания на топлинния поток, получавани от някой датчик за топлина. Както виждате, графиката показва наличието на силна линейна зависимост между тези две променливи.

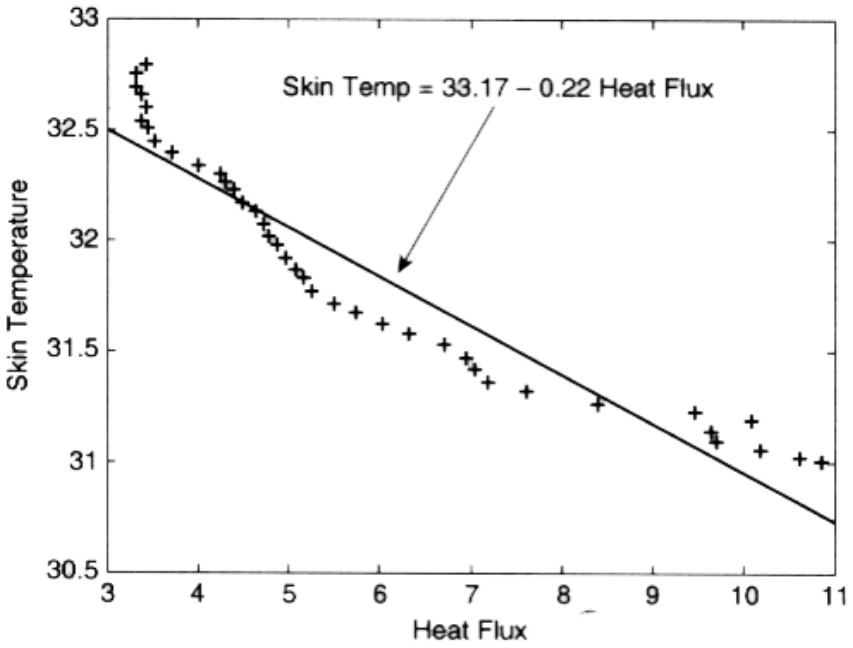
Използвайки данните от примера получаваме, че $\sum_{i=1}^{39} x_i = 229.9, \sum_{i=1}^{39} x_i^2 = 1559.2, \sum_{i=1}^{39} y_i = 1242.9, \sum_{i=1}^{39} x_i y_i = 7279.7,$ следователно за да получим търсените стойности на параметрите, нормалното уравнение може да бъде решено по следния начин:

$$\begin{pmatrix} \beta_1 \\ \beta_0 \end{pmatrix} = \begin{pmatrix} 39 & 229.9 \\ 229.9 & 1569.2 \end{pmatrix}^{-1} \begin{pmatrix} 1242.9 \\ 7279.7 \end{pmatrix} = \begin{pmatrix} 0.1881 & -0.0276 \\ -0.0276 & 0.0047 \end{pmatrix} \begin{pmatrix} 1242.9 \\ 7279.7 \end{pmatrix} = \begin{pmatrix} 33.1699 \\ -0.2208 \end{pmatrix}$$

Следователно, линейният модел, който най-добре приляга към данните в термините на минимизацията на квадратичната грешка е:

*r(**β**, x) = -0.22x + 33.17*

Графиката на този модел е показана н фиг. 9.2.



Фиг. 9-2. Линейния модел, отговарящ на данни от фиг. 9.1.

Може да се покаже, че общото решение на нормалното уравнение (9.8) може да се изрази по следния начин:

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$
$$\beta_1 = \frac{\sigma_{xy}}{\sigma_{xx}} \qquad (9.9)$$

където:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,\ \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

$$\sigma_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \quad (9.10)$$

$$\sigma_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

По този начин, моделът, който осигурява минимума на квадратичната грешка, се задава по следния начин:

$$r(x) = \bar{y} + \frac{\sigma_{xy}}{\sigma_{xx}}(x - \bar{x}) \quad (9.11)$$

Множествена линейна регресия

Множествената линейна регресия е директно обобщение на проста линейна регресия за случай с *p* променливи, т.е.

$$r(\boldsymbol{\beta}, \mathbf{x}_i) = \beta_0 + \beta_1 x_{1i} + ... + \beta_p x_{pi} \quad (9.12)$$

И в този случай моделът се получава с използване на критерия за най-малка квадратична грешка – той се опитва на намери вектор от параметрите **β**, който минимизира сума на квадрати на грешките:

$$LSE = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{1i} + ... + \beta_p x_{pi}))^2 \quad (9.13)$$

Ако въведем обозначение:

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & ... & x_{1p} \\ 1 & x_{11} & x_{12} & ... & x_{1p} \\ ... & ... & & & \\ 1 & x_{11} & x_{12} & ... & x_{1p} \\ & & & & \end{pmatrix} \quad (9.14)$$

То нормалното уравнение (9.8) може да се препише в матричната форма:

$$\mathbf{X^T X}\beta = \mathbf{X^T y} \quad (9.15)$$

Едно ефективно решение на това уравнение е:

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

където $\mathbf{X}^T$ означава транспонирана матрица на **X**, а $\mathbf{X}^{-1}$ - обратната матрица.

Тъй като обратната матрица не винаги съществува, този подход страда от числова нестабилност. Една по-добра алтернатива се състои в използването на техника, наречена декомпозиция на сингуларните стойности (SVD – Singular

Value Decomposition), която се използва за намиране на решение на системите на уравнение във вид $\mathbf{X\beta} = \mathbf{y}$.

Много нелинейни регресионни модели (когато целевата функция е например полином със степен по-голяма от 1 от някои от променливите) могат да бъдат сведени към линейната регресия чрез въвеждане на нови променливи.

Например, нелинейният модел $y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_3^3$ може да се сведе към линейния: $y = \beta_0 + \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_3$ чрез замени: $y_1 = x_1$, $y_2 = x_1^2$, $y_3 = x_3^3$.

9.3.1.2. Непараметрични подходи (локално моделиране)

Основната идея на непараметрична (локална) регресия се състои в получаване на предсказване за някоя точка $\mathbf{x}_q$ чрез апроксимация на някоя параметрична функция *само в околността* на $\mathbf{x}_q$. Изследванията върху локалната регресия водят своето начало още от 19-й век. Съвременните работи върху локалното моделиране започват през 50-ти години на 20 век с разработка на така наречени *методи на ядра (kernel methods)*. При този подход предсказването върху тестовия пример се получава чрез осредняването на Y стойностите на обучаващи примери, попаднали в околността с размер h около тестовия пример. Параметър h се нарича *ширина на лента (bandwidth)*. Всеки обучаващ пример, попаднал в околността на тестовия пример, участва в предсказването на неизвестната стойност с различно тегло, зависещо от разстояние от него до тестовия пример. Тази изчислителна схема се реализира чрез така наречена *функция на ядра* или *тегловната функция*. Съществуват множество различни тегловни функции, като една от най-често използвани е ядрото на Гаус. Ако чрез d означим мярка, изчисляваща разстояние между два примера (най-често това е претеглено Евклидово разстояние $d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{p} w_k (x_{ik} - x_{jk})^2}$, където w_i са тегло на атрибут i, то ядрото на Гаус има вид:

$$K(d) = e^{-d^2}$$

Предсказването (локален регресионен модел) в тестовата точка $\mathbf{x}_q$ се получава от следния израз:

$$r(\mathbf{x}_q) = \frac{\sum_{i \in H(\mathbf{x}_q, h)} K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right) \times y_i}{\sum_{i \in H(\mathbf{x}_q, h)} K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right)}, \text{ където } H(\mathbf{x}_q, h) = \{\mathbf{x}_i \mid d(\mathbf{x}_i, \mathbf{x}_q) \le h\}$$

Обърнете внимание, че когато ширината на лента h се избира като разстояние до k-я най-близък съсед, а функцията на ядра е постоянна (т.е. дава едно и също тегло на всички обучаващи примери в пределите на лентата), то описаният метод съвпада с вече изученият от нас метод на k-най-близки съседи, използван в машинно самообучение и разпознаване на образи.

Локалната полиномна регресия е едно обобщение на ранните работи върху описаните по-горе методи на ядра. На практика, регресията на ядра

представлява апроксимация чрез полином от нулева степен (т.е. константа) в околността на тестовия пример. В общия случай целта на локална полиномна регресия е да направи апроксимация на целевата функция чрез полином от степен *m* в околността на тестовата точка $\mathbf{x}_q$ чрез обучаващи примери от околността на тази точка. Интересни резултати са получени за *локална линейна регресия* (*m = 1*), която може да се разглежда като опит за прилагане на (глобален) модел на линейната регресия в контекста на локалното моделиране.

Основната разлика на локална линейна регресия от своя глобален вариант е, че с всяка обучаваща точка е асоциирано определено тегло, което е функцията от разстояние между тази точка и тестовата точка. За да намерим апроксимацията чрез локален линеен полином в околностите на $\mathbf{x}_q$, трябва да намерим параметрите на модела, минимизиращи критерий:

$$\sum_{i=1}^{n} (y_i - \boldsymbol{\beta}\mathbf{x}_i)^2 K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right)$$

Може да се покаже, че намирането на параметри **β**, минимизиращи указания критерий, се свежда до решение на матричното уравнение:

$$(\mathbf{Z}^T\mathbf{Z})\boldsymbol{\beta} = \mathbf{Z}^T\mathbf{q},$$

където **Z** = **VX**, а **q** = **Vy**.

Матрица **V** е диагоналната матрица $\mathbf{V} = diag(v_1, v_2, \ldots, v_n)$, където

$$v_i = \sqrt{K\left(\frac{d(\mathbf{x}_{i,}\mathbf{x}_q)}{h}\right)}$$

Полученото матрично уравнение има същия формат, като този за множествената линейна регресия, което означава, че то може да бъде решени с използване на същите изчислителни процедури (обръщането на матрицата **Z** или прилагане на SVD метод).

Един от основни проблеми при използването на методите за локално моделиране е, че при голяма размерност на входното пространство (голям брой атрибути) локалната околност вече трудно да се разглежда като действително локална. Друг проблем е липсва на разбираемостта на локалните модели.

9.3.1.3. Полу-параметрични модели

Основната идея на полу-параметрични модели е интегрирането в един общ модел както на локални (непараметрични), така и на глобални (параметрични) компоненти. Например, *частични линейни модели* интегрират линейната компонента с модела на ядра. Един частично линеен модел може да се опише чрез **y** = **βx** + m(**x**), където **βx** е линеен модел с параметрите **β**, минимизиращи най-малките квадрати на грешките, а m(**x**) – е регресионен модел на ядра.

Един от начините за използване на такива модели се състои в първоначалното генериране на параметричната компонента на модела (т.е. създаване на глобален линеен полином), а след това да бъдат приложени регресията на ядра към остатъци (грешките) на този модел. Следвайки този подход създаването на модела се започва с изчисляване на линейния модел (например чрез прилагане на техниката SVD). След това за дадена тестова точка се определят обучаващи точки, намиращи в указаната ширина на лентата (околността). За всяка от тези точки се изчисляват грешките на линейния модел:

$$r_i = \boldsymbol{\beta}\mathbf{x}_i - y_i, < \mathbf{x}_i, y_i > \in H(\mathbf{x}_q, h)$$

Тези грешки се използват за получаване на тяхно претеглено локално предсказване в околността на тестова точка. И накрая, предсказаните грешки се добавят към предсказването на линейния модел за тестовата точка, образувайки предсказване на частично линеен модел:

$$r(\mathbf{x}_q) = \boldsymbol{\beta}\mathbf{x}_q - \frac{\sum_{i \in H(\mathbf{x}_q, h)} K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right) \times (\boldsymbol{\beta}\mathbf{x}_i - y_i)}{\sum_{i \in H(\mathbf{x}_q, h)} K\left(\frac{d(\mathbf{x}_i, \mathbf{x}_q)}{h}\right)}, \text{ където } H(\mathbf{x}_q, h) = \{\mathbf{x}_i \mid d(\mathbf{x}_i, \mathbf{x}_q) \le h\}$$

Тези модели могат да се разглеждат като правещи определени локални "корекции" на предсказвания на глобалния линеен модел, представени чрез $\boldsymbol{\beta}\mathbf{x}_q$.

9.3.2. Методи от машинното самообучение

Изкуствени невронни мрежи

Основни алгоритми за обучение на изкуствени невронни мрежи бяха разгледани в курса по машинно самообучение. Общата схема на прилагането на невронни мрежи за решаване на регресионната задача се състои в използване на три-слойна невронна мрежа с един скрит слой. Възлите от входния и скрит слой са сигмоидни възли, докато изходният възел се представя чрез безпраговият възел. За обучението на такава мрежа се използва алгоритъм BACKPROPAGATION.

Основните критики към невронните мрежи са тяхната бавна сходимост към решението и сложността на интерпретацията на намереното решение от потребители.

Самообучение чрез запомняне

Методите, базирани на алгоритъма на k-най-близки съседи са подобни на вече разгледани подходи за локалното моделиране. При използването на тези методи за предсказване могат да бъдат използвани и други тегловни схеми, отлични от разгледаните в раздела за моделиране чрез ядра, обаче всички те се базират върху разстояния между най-близките съседи и тестовата точка. Те могат да бъдат използвани и глобално (например методът на Шепард) чрез прилагане на същата тегловна схема към всички обучаващи примери.

Основният проблем с тези методи е тяхната зависимост от нерелевантни атрибути и сложността за интерпретация от потребителя на предложеното решение.

Регресионни правила

Съществуват няколко системи за научаване на регресионни правила, описани на пропозиционален език (например CUBIST (Quinlan 1993)). Условната част на тези правила са конюнкция от тестове върху входните атрибути, а заключителната част съдържа предсказване за стойността на целевия атрибут. Правилата могат да дават предсказване във вид на средната стойност на целевите стойности на случаи, покрити от условната част на правило. Съществуват и варианти, в които предсказването се прави чрез използването на алгоритъма на k-най-близки съседи, избрани от обучаващите примери, покрити от условната част на правилото. Правилата се научават чрез трансформирането на регресионната задача в задачата за класификация. Практически, стойностите на целевият атрибут се дискретизират и медианите на дискретизационни интервали се използват като имената на класове. След това правилата се научават чрез прилагането на разделящи или покриващи стратегии (виж за подробности курса по МС). Името на класа (медианната стойност) се използва за предсказване в дясната част на правило, или към покрити от правилото обучаващи примери се прилагат други регресионни модели (както параметрични, така и непараметрични).

Регресионни дървета

Използването на дървовидни модели за решаване на регресионни задачи е един от най-интензивно развиващи се подходи в машинно самообучение. В тази и следващата лекции ще спрем по-подробно върху методи за научаване на такива дървета, методи за тяхното подрязване, както и на комбинирането им с локални регресионни модели.

9.4. Древовидни модели

Едно регресионно дърво може да се разглежда като един вид адитивен модел във вида:

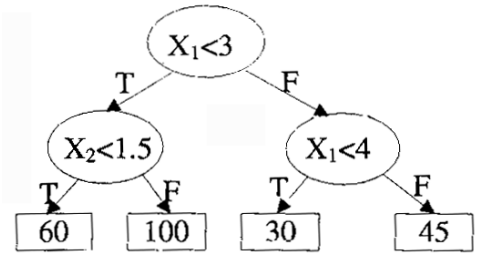
$$r(\mathbf{x}) = \sum_{i=1}^{l} k_i \times \delta(\mathbf{x} \in D_i) \qquad (9.16)$$

където

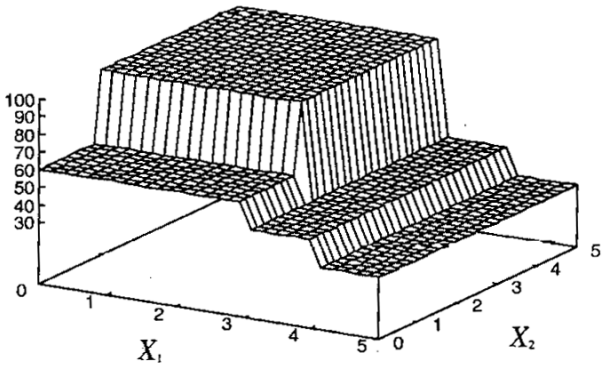
k_i са константи;
$\delta(.)$ е индикаторната функция, връщаща 1, ако нейния аргумент е истина и 0 – ако лъжа, а
D_i са непресичащи се области от обучаващите данни, такива, че

$$\bigcup_{i=1}^{l} D_i = D, \text{ и } \bigcap_{i=1}^{1} D_i = \varnothing$$

Моделите от този тип понякога се наричат частично-постоянни (piecewise constant) регресионни модели, тъй като те разбиват пространството на атрибути на едно множество от области, като във всяка от тях използва една постоянна величина за апроксимацията на целевия атрибут. Важен аспект на тези модели е, че те позволяват представянето на тези области със средства на пропозиционалната логика във вид на дърво. На всеки път в дървото от корена до листо съответства една област. Всяко листо съдържа стойността на целевия атрибути (или на някоя функция, използвана за неговото изчисляване). Всеки междинен (вътрешен) възел на дървото е логическият тест върху някой атрибут. В частност в двоични дървета всеки тест има само два изхода – истина и лъжа. Това означава, че със всяка област D_i се асоциира път P_i, който се състои от конюнкция от логическите тестове върху атрибутите. Това символно представяне на регресионната функция е много важно за по-добро разбиране на регресионната повърхност.



Фиг. 9-3. Пример на двоично регресионно дърво



Фиг. 9-4. Регресионната повърхност, съответстваща на дървото

На Фиг. 9-3 е показан един пример на регресионното дърво. Тъй като в него съществуват четири различни пътя от корена до листата, то това дърво разбива входното пространство на четири различни области. Дървото съответства на регресионната повърхност, показана на Фиг. 9-4. Регресионният модел на дървото има вид:

$$r(\mathbf{x}) = 60 \times \delta(X_1 < 3 \wedge X_2 < 1.5) + 100 \times \delta(X_1 < 3 \wedge X_2 \geq 1.5) +$$
$$30 \times \delta(X_1 \geq 3 \wedge X_2 < 4) + 45 \times \delta(X_1 \geq 3 \wedge X_2 \geq 4)$$

Регресионните дървета се създават чрез използване на алгоритъма за рекурсивно разделяне (подобен на този, за класификационните дървета). Той построява дървото чрез рекурсивното разбиване на обучаващите примери на по-малки подмножества. На Фиг. 9-5 е приведено общото описание на този алгоритъм. Като вход алгоритъмът получава множеството от обучаващи примери $D_t = \{<\mathbf{x}_i, y_i>\}$, $i = 1, \ldots, n$, и ако зададеният критерий за край не е удовлетворен, той генерира тестовия възел t, чиито клонове се получават чрез прилагането на същия алгоритъм към двата подмножества от примерите.

Тези подмножества се състоят от случаите, които удовлетворяват теста за разделяне s^* във възела t: $D_{t_L} = \{< \mathbf{x}_i, y_i > \in D_t : \mathbf{x}_i \rightarrow s^*\}$, и останалите случаи

$D_{t_R} = \{< \mathbf{x}_i, y_i > \in D_t : \mathbf{x}_i \not\rightarrow s^*\}$. Във всеки възел се избира най-добрият тест за разделяне съгласно зададения локален критерий.

Вход: Множество обучаващи примери $D_t = \{<\mathbf{x}_i, y_i>\}$, $i = 1, ..., n$
Изход: Регресионно дърво

АКО *Критерий за край* ***ТО***
Създай *Възел-листо* и назначи му някоя *Постоянна стойност*
Върни *Възел-листо*
ИНАЧЕ
Намери *Най-добрия разделящ тест s**
Създай *Възел t* с *s**
Ляв_клон(t) = Алгоритъм_за_рекурсивно_разделяне($\{<\mathbf{x}_i, y_i > : \mathbf{x}_i \rightarrow s^*\}$)
Десен_клон(t) = Алгоритъм_за_рекурсивно_разделяне($\{<\mathbf{x}_i, y_i > : \mathbf{x}_i \not\rightarrow s^*\}$)
Върни *Възел t*
КРАЙ

Фиг. 9-5. Алгоритъм за създаване на регресионното дърво чрез рекурсивно разделяне

Алгоритъмът има три основни компонента:

- Начин за избиране на най-добрият разделящ тест (разделящото правило)
- Правило за определяне на спиране на нарастване на дървото (критерий за край)
- Правило за назначаване на стойността на целевия атрибут във всеки терминален възел (листо)

В следващите раздели ще разгледаме най-често срещаните подходи към решаване на тези проблеми.

9.4.1. Регресионни дървета на най-малките квадрати (LS – регресионни дървета)

Най-често използваният метод за построяване на регресионните модели се състои в търсенето на параметрите на модела, които минимизират *средна квадратична грешка* (MSE – Mean Squared Error):

$$MSE(r) = \frac{1}{n}\sum_{i=1}^{n}(y_i - r(\boldsymbol{\beta}, \mathbf{x}_i))^2 \quad (9.17)$$

Такива методите се наричат *регресионни методи на най-малки квадрати* (Least Squares Regression). Едни от най-известни системи, реализиращи такива методи са RETIS (Karalic and Cestnik 1991), M5 (Quinlan 1992) и CART (Breiman et al. 1984).

За MSE-критерий е доказана следната теорема:

Константа k, минимизираща средна квадратична грешка, е средната стойност на целевият атрибут.

Съгласно тази теорема, константата, която трябва да бъда назначавана в листата на някое регресионно дърво, построено с използване на критерия за минимизиране на средна квадратична грешка, е средната стойност на целевия атрибут в обучаващите примери, покрити от съответното листо *l*:

$$k_l = \frac{1}{n_l}\sum_{D_l} y_i \quad (9.18)$$

където n_l е размер на множеството D_l, съдържащо примерите, покрити от листо *l*.

При извода на разделящото правило ще се ограничим с двоични регресионни дървета (които са най-често срещания случай). Разделящият тест се избира с цел да намали грешката при апроксимацията на полученото дърво. Считайки, че константната стойност на целевият атрибут, назначавана във възела t на дървото, се избира съгласно формулата (9.18), ще дефинираме апроксимиращата грешка на дървото във възела *t* като средно от квадратите на разликите между стойностите *y* на примерите, покрити от възела и съответната константа на възела k_t:

$$Err(t) = \frac{1}{n_t}\sum_{D_t}(y_i - k_t)^2 \qquad (9.19)$$

Ще дефинираме и грешката на дърво *T* като претеглената средна стойност на грешките в листата на дървото:

$$Err(T) = \sum_{l\in\widetilde{T}} P(l)\times Err(l) = \sum_{l\in\widetilde{T}} \frac{n_l}{n}\times\frac{1}{n_l}\sum_{D_t}(y_i - k_l)^2 = \frac{1}{n}\sum_{l\in\widetilde{T}}\sum_{D_t}(y_i - k_l)^2 \qquad (9.20)$$

където:

$P(l)$ е вероятността, че един пример попада в корен *l*;
$\widetilde{T}$ е множеството от листата на дървото *T*.

Едно двоично разделяне разбива множеството от примери на две части, като целта е максимално да намали грешката на дървото, получавано при това разделяне. Ще дефинираме грешката от разделяне *s* като претеглено средно от грешките в получаваните под-възли:

$$Err(s,t) = \frac{n_{t_L}}{n_t}\times Err(t_L) + \frac{n_{t_R}}{n_t}\times Err(t_R) \quad (9.21)$$

където индексите *L* и *R* се използват за означаване на левият наследник на текущ възел (съдържащ примери, удовлетворяващи теста s) и неговият десен наследник.

Като най-доброто разделяне s* ще дефинираме такъв тест, който максимизира разликата между грешките на дърво преди и след разделянето:

$$s^* = \arg\max_s \Delta Err(s,t) = \arg\max_s (Err(t) - Err(s,t)) \quad (9.22)$$

И така, на всяка итерация алгоритъмът за рекурсивно разделяне ще оценява всички възможни разделяния по всеки от атрибути и избира този, който максимизира разликата *ΔErr*.

Последният аспект на разглежданият алгоритъм за пораждане на регресионни дървета е свързан с критерия за спиране. Основният проблем е свързан с надеждността на оценките за грешки, използвани при избора на разделящите тестове. Всички използвани оценки са функции на обучаващата извадка (т.е. извадкови грешки). Точността на тези оценки силно зависи от качеството на извадката. Тъй като алгоритъмът рекурсивно разделя първоначалното множество от примери на все по-малки и по-малки подмножества, то точността на използваните оценки намалява с нарастването на дървото. Може да се докаже, че при построяване на дървото стойността на *ΔErr* винаги е по-голяма или равна на нула, т.е. от тази гледна точка ние получаваме все по-точен и по-точен регресионен модел. В екстремален случай едно извънредно голямо дърво, покриващо само по един случай във всяко свое листо, ще има нулевата грешка. Проблемът с тези разсъждения е свързан с надеждността на оценки, базирани на обема на обучаващите данни, от които те са изведени. Оценките, получени върху малки извадки, са ненадеждни и водят до създаване на модели с малка предсказваща точност. Този проблем ни вече е известен под името *преспецификация* на дървото към обучаващите данни.

За намаляване на ефекта на този проблем са възможни два подхода. Първият се състои в дефиниране на някой надежден критерий, определящ кога трябва да бъде спряно нарастването на дървото. В контекста на дървовидни модели този подход се нарича *предварително подрязване (pre-pruning)*. Вторият, по-често прилаган подход, се състои в създаване на едно много голямо (и ненадеждно) дърво, което след това се подрязва – *завършващото подрязване* (post-pruning). Подрязването на регресионни дървета е много важна стъпка за получаване на точни модели и ще бъде разгледан в следващата лекция. При използване на предварителното подрязване един често използван критерий за спиране на нарастването на дървото се състои в указване на едно минимално количество от примери, което, ако бъде достигнато, води до спиране на работата на алгоритъма. Другият вариант на критерия за спиране е правилото, при което листото се създава, ако грешката в текущ възел е под зададения процент (или част) от грешката във корневият възел.

9.4.2. Ефективно създаване на LS-регресионни дървета

Изчислителната сложност на рекурсивно разделящият алгоритъм, използван за създаване на регресионни дървета, е силно зависи от избора на най-добрия тест за дадения възел. Броят на възможни разделяния зависи от типа на тествания атрибут. На Фиг. 9-6 е приведен един по-детайлен вариант на нашия алгоритъм:

Вход: Множество от *n* обучаващи примери $D_t = \{<\mathbf{x}_i, y_i>\}, i = 1, ..., n$
Изход: Регресионно дърво

АКО критерий за край ***ТО***
Създай *Възел-листо* и назначи му средното на *Y* стойностите на примери, покрити от *Възел-листо*
Върни *Възел-листо*
ИНАЧЕ
Нека *s**=<*произволно разделяне*>
ЗА всички атрибути X_V ***НАПРАВИ***
АКО X_V е номинален атрибут ***ТО***
Най-доброто_разделянеX$_V$ =
Опитай_всички_номинални_разделяния({<$\mathbf{x}_i$, y_i>}, X_V)
ИНАЧЕ АКО X_V е непрекъснат атрибут ***ТО***
Най-доброто_разделянеX$_V$ =
Опитай_всички_непрекъснати_разделяния({<$\mathbf{x}_i$, y_i>}, X_V)
КРАЙАКО
АКО Най-доброто_разделянеX$_V$ е по-добро от s* ***ТО***
s = Най-доброто_разделянеX*$_V$
КРАЙАКО
КРАЙЗА
Създай *Възел t* с *s**
Ляв_клон(t) = Алгоритъм_за_създаване_на_LS-дърво($\{<\mathbf{x}_i, y_i> : \mathbf{x}_i \rightarrow s^*\}$)
Десен_клон(t) = Алгоритъм_за_създаване_на_LS-дърво ($\{<\mathbf{x}_i, y_i> : \mathbf{x}_i \not\rightarrow s^*\}$)
Върни *Възел t*
КРАЙ

Фиг. 9-6. Алгоритъм за създаване на LS-регресионно дърво.

Основната маса от изчисления е свързана с изчисляване на всички възможни разделяния. Предполагайки постоянен регресионен модел в листата (ф-ла (9.18)), за оценяване на всеки тест (ф-ла (9.22)) за всеки клон трябва да изчисляваме две средни стойности (ф-ли (9.19) и (9.20)). На практика ф-ла (9.19) е същата, като и формула за изчисляване на дисперсията на една случайна променлива. Тези изчисления изискват двукратно минаване през данни: първото, за да изчисли средна стойност, а второто – за да изчисли квадрати от разликите. Обемът на тези изчисления може да бъде намален чрез използване на еквивалентната формула, позволяваща само еднократно минаване през данни:

$$Err(t) = \frac{\sum_{D_t} y_i^2}{n_t} - \left(\frac{\sum_{D_t} y_i}{n_t} \right)^2 \qquad (9.23)$$

Обаче, дори при използването на тази формула цената на оценяването на всяко възможно разделяне ще бъде $O(n_t)$. Луис Торго (Torgo 1999) предлага един по-ефективен начин, позволяващ инкрементално оценяване на всички възможни разделяния на атрибута.

Подставяйки ф-ла (9.23) в ф-ла (9.21) грешка при разделянето *Err(s, t)* може да бъде представена по следния начин:

$$Err(s,t) = \frac{1}{n_t}(SS_L + SS_R) - \frac{1}{n_t}\left(\frac{S_L^2}{n_{t_L}} + \frac{S_R^2}{n_{t_R}} \right) \qquad (9.24)$$

където:

$$SS_L = \sum_{D_{t_L}} y_i^2,\ SS_R = \sum_{D_{t_R}} y_i^2,\ S_L = \sum_{D_{t_L}} y_i \text{ и } S_R = \sum_{D_{t_R}} y_i$$

Лесно е да се види, че първият член в ф-ла (9.24) е константа за всички оценявани разделяния, тъй като:

$D_t = D_{t_L} \cup D_{t_R}$ и, следователно, $\sum_{D_{t_L}} y_i^2 + \sum_{D_{t_R}} y_i^2 = \sum_{D_t} y_i^2$.

Това означава, единствената разлика между различни кандидати за разделяне е вторият член от ф-ла (9.24).

Това опростяване води до новата дефиниция на най-доброто разделяне във възела:

Най-доброто разделяне s* е разделянето, което максимизира израз:

$$\frac{S_L^2}{n_{t_L}} + \frac{S_R^2}{n_{t_R}}, \text{ където } S_L = \sum_{D_{t_L}} y_i \text{ и } S_R = \sum_{D_{t_R}} y_i \quad (9.25)$$

Тази дефиниция позволява реализацията на бърза инкрементална процедура по оценяване на всички кандидати за разделяне на всеки атрибут, която ще разгледаме в следващите раздели.

Разделяне по непрекъснати атрибути

На Фиг. 9-7 е показан алгоритъм за намиране на най-доброто разделяне за един непрекъснат атрибут, базиран на дефиницията (10).

***Вход*:** n_t обучаващи примера, сума на техните Y стойности (S_t), атрибут X_V.
***Изход*:** Най-добата точка на разделяне по X_V.

Сортирай примери съгласно техните стойности по X_V
$S_R = S_t$; $S_L = 0$
$n_R = n_t$; $n_L = 0$
Най-добра_досега = 0
ЗА всички примери *i* ***НАПРАВИ***
    $S_L = S_L + y_i$; $S_R = S_R - y_i$
    $n_L = n_L + 1$; $n_R = n_R - 1$
    АКО $(X_{i+1,V} > X_{i,V})$ ***ТО***
        НоваСтойност_на_Разделяне = (S_L^2 / n_l) + (S_R^2 / n_R)
        АКО *(НоваСтойност_на_Разделяне > Най-добра_досега)* ***ТО***
            Най-добра_досега = НоваСтойност_на_Разделяне
            Най-добър_праг = ($X_{i+1,V} + X_{i,V}$) / 2
        КРАЙАКО
    КРАЙАКО
КРАЙЗА

Фиг. 9-7. Алгоритъм за намиране на най-доброто разделяне на непрекъснат атрибут

Алгоритъмът се състои от две основни части. Първата сортира примерите по техните стойности на разглеждания атрибут и има средната сложност $O(n_t \log$

n_t) при използване на алгоритъма за сортиране QuickSort. Сортирането е необходимо, за да могат да бъдат проверени всички възможни прагове, разбиващи множеството от стойности на атрибута на две части (< и ≥ от прага). Втората част на алгоритъма оценява всички кандидати за разделяне. Броят на възможни прагове е не повече от $n_t - 1$ (ако всички примери имат различни стойности на проверявания атрибут). Следователно, изчислителната сложност на тази операция е $O(n_t - 1)$

Разделяне по номинални атрибути

Разделянето по някой номинален (дискретен) атрибут обикновено включва проверка на всички възможни тестове от вида на $X_V = x_v$, където x_v е една от възможните стойности на атрибута X_V. Големият брой от възможните стойности води до увеличаване на размера на дървото. Една възможна алтернатива е използването не на двоични дървета, а да имаме по един клон за всяка възможна стойност. Проблемът с този подход е бързото намаляване на обучаващите примери по всеки от клоновете, което води до ненадеждни оценки за стойностите на грешки, определящи развитието на дървото. Още една възможна алтернатива е да разглеждат тестове във вид $X_V \in \{x_{v_1}, ..., x_{v_k}\}$. Макар че този подход води до допълнителна изчислителна сложност, той позволява да подобри разбираемостта на получените дървета и не разделя прекалено силно обучаващите примери. Breiman et al. (1984) са доказали теоремата, която променя сложността за получаване на този тип разделяния от $O(2^{\#X_v - 1})$ на $O(\#X_v - 1)$, където $\#X_v$ е брой на възможни стойности на съответния атрибут. Предложеният от Breiman et al. метод включва първоначалното сортиране на примери във възела по следния начин. Да означим чрез $B=\{b_i\}$ - множеството от възможни стойности на атрибута X_v в текущия възел t и да дефинираме $\bar{y}(b_i)$ като средното на Y стойностите на примерите, за които $X_v = b_i$, тогава можем да ги сортираме по такъв начин, че: $\bar{y}(b_1) \le \bar{y}(b_2) \le ... \bar{y}(b_{\#B})$. Breiman и неговите колеги доказаха теоремата, че:

Най-доброто разбиване на номиналния атрибут X във възела t е едно от следните $\#B - 1$ разбивания:

$$X_v \in \{b_1, b_2, ..., b_h\}, h = 1, ..., \#B - 1$$

За по-голяма яснота да разгледаме следния пример: да предположим, че имаме следните случаи в някой възел t:

ЦВЯТ	**...**	**Y**
Зелен		24
Червен	...	56
Зелен	...	29
Зелен	...	13
Син	...	120
Червен	...	45
Син	...	100

Получаваме следните средни стойности:

$\bar{y}$ *(Зелен) = (24 + 29 + 13) / 3 = 22*

$\bar{y}$ *(Червен) = (56 + 45) / 2 = 50.5*

$\bar{y}$ *(Син) = (120 + 100) / 2 = 110*

Следователно, ако сортираме стойностите на номинален атрибут *Цвят* съгласно средните стойности на *Y*, то получим следното подреждане <*Зелен, Червен, Син*>. Тогава, съгласно теоремата на Breiman най-доброто разбиване ще бъде едно от следните 2 = 3-1 разбивания: $X_v \in \{Зелен\}$ *и* $X_v \in \{Зелен, Червен\}$.

И така, имайки сортираните примери, съгласно описания по-горе метод, можем да използваме следния инкрементален алгоритъм за намиране на най-доброто разделяне, подобен на този за непрекъснатия атрибут:

***Вход*:** n_t обучаващи примера, сума на техните *Y* стойности (S_t) и атрибут X_V.
***Изход*:** Подредено множество от стойностите на X_V и най-доброто разделяне на това множество

Изчисли средните *Y* стойности, асоциирани с всяка стойност на X_V
Сортирай стойностите на X_V съгласно асоциираните с тях средните *Y* стойности
$S_R = S_t$; $S_L = 0$
$n_R = n_t$; $n_L = 0$
Най-добра_досега = 0
ЗА всяка стойност *b* от сортираното множество ***НАПРАВИ***
    YB = сума на *Y* стойности на примери с $X_v = b$
    NB = броят на примери с $X_v = b$
    $S_L = S_L + YB$; $S_R = S_R - YB$
    $n_L = n_L + NB$; $n_R = n_R - NB$
    НоваСтойност_на_Разделяне = $(S_L^2 / n_l) + (S_R^2 / n_R)$
    АКО *(НоваСтойност_на_Разделяне > Най-добра_досега)* ***ТО***
        Най-добра_досега = НоваСтойност_на_Разделяне
        Най-добър_праг = позиция на b в сортираното множество от стойности
    КРАЙАКО
КРАЙЗА

Фиг. 9-8. Алгоритъм за намиране на най-доброто разделяне на номинален атрибут

Изчислителната сложност на този алгоритъм е по-малка в сравнение със случая на непрекъснатия атрибут. На практика тя се определя от броя на стойности на атрибута *(#B)*. Изключението е частта, сортираща стойности съгласно техните средни *Y* стойности. Процедурата за сортиране е *O(#B log #B)*, но получаване на средните *Y* стойности, асоциирани с всяка стойност *b* изисква минаване през всички примери $O(n_t)$, което в общия случай е с по-голяма сложност.