Системи, основани на знания - зимен семестър,
2022/2023 учебна година

Тема 9:
Структурни формализми за представяне и използване на знания

Работа със знания за света в СОЗ

Знанията на човека за света (битовите знания, commonsense knowledge) обхващат много области, чието формално описание е много трудно поради редица причини. Границата между битовите („наивните“) и специализираните (експертните) знания е размита и зависи от нивото, необходимо за осъществяването на всяка конкретна дейност.

Характерни особености на битовите знания:

- *голям обем*. Полезни експертни системи могат да се изградят с използване на факти и правила, чийто брой е от няколкостотин до няколко хиляди. За описание на „наивните“ знания на човека за света са необходими между един и десет милиона факта (такъв е броят на фактите от разглеждания вид в онтологията Cyc);
- *липса на определени външни и вътрешни граници*. Не е лесно да се каже нито какво точно включват битовите знания, нито как е целесъобразно да се разделят на части, които могат да се описват и изучават поотделно;

- *трудност при представянето на някои категории, чужди на традиционната логика* (например минало и бъдеще време, условност, намерение и др.);
- *голям дял на приблизителните твърдения*, с които традиционната логика не може да се справи.

Необходимост от работа с битови знания в СОЗ:

- системи, разполагащи с битови знания, биха били приложими в ежедневната човешка дейност;
- експертните системи биха били по-полезни, ако имаха повече знания за начина на мислене на потребителите си (който е човешки, следователно битов);
- разширяването на знанията на експертните системи вероятно би било по-ефективно, ако те познаваха основните аналогии и метафори на човешкото мислене;
- разбирането на естествен език по принцип изисква битови знания.

Представяне и използване на знания чрез семантични мрежи

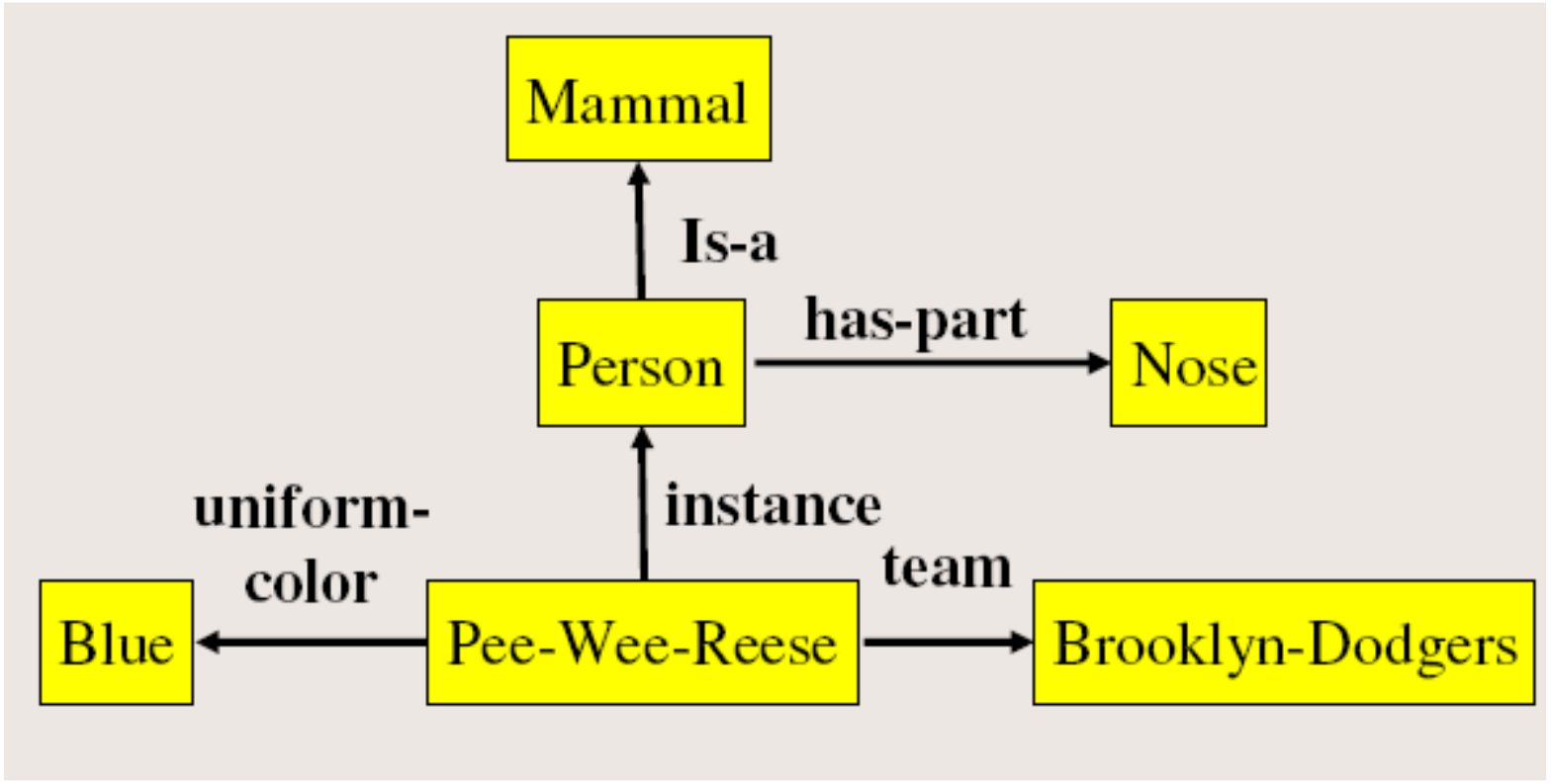
Обща характеристика. Декларативен формализъм, един от т. нар. структурни формализми за ПИЗ (при структурните формализми знанията се представят чрез множество структурни единици (елементи), свързани помежду си чрез явно зададени връзки).

Семантичните мрежи (СМ) представляват ориентирани графи, формулиращи таксономична информация за предмети или обекти (вкл. опредметени абстрактни понятия) и техните свойства.

С възлите в една СМ се означават обектите (понятията, класовете и т.н.) от предметната област, а с дъгите – връзките (отношенията, релациите) между тях.

Първоначално СМ са използвани за описание (представяне) на значението на думи от компютърни речници.

Пример


Mammal
Is-a
Person
has-part
Nose
uniform-
color
instance
team
Blue
Pee-Wee-Reese
Brooklyn-Dodgers

Типове възли в СМ:

- релационни константи (таксономични категории или свойства);
- обектни константи (предмети или обекти от областта).

Типове дъги в СМ:

- тип „подмножество“ (описват релации от тип клас – суперклас);
- тип „елемент“ (описват релации от тип обект – клас);
- тип „функция“ (описват свойства на обектите и класовете).

Използване на знанията, представени чрез СМ:
наследяване (немонотонни разсъждения).

Разширяване на изразителната сила на семантичните мрежи:

- Предствяне на произволни n-арни релации: метод на Саймънс;
- Представяне на знания, които се формулират чрез твърдения, съдържащи квантори за съществуване и всеобщност: разделени СМ (partitioned semantic nets).

Представяне на унарни релации: трансформират се в бинарни.

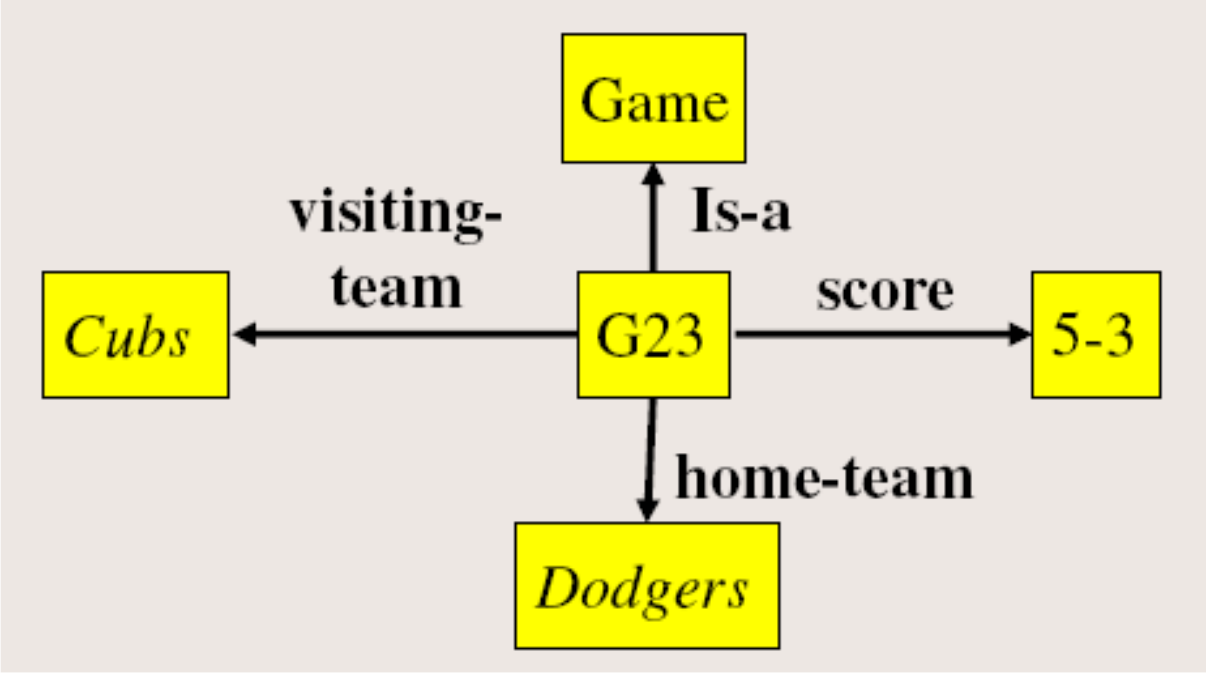
Например,

man(Marcus)

може да се трансформира в

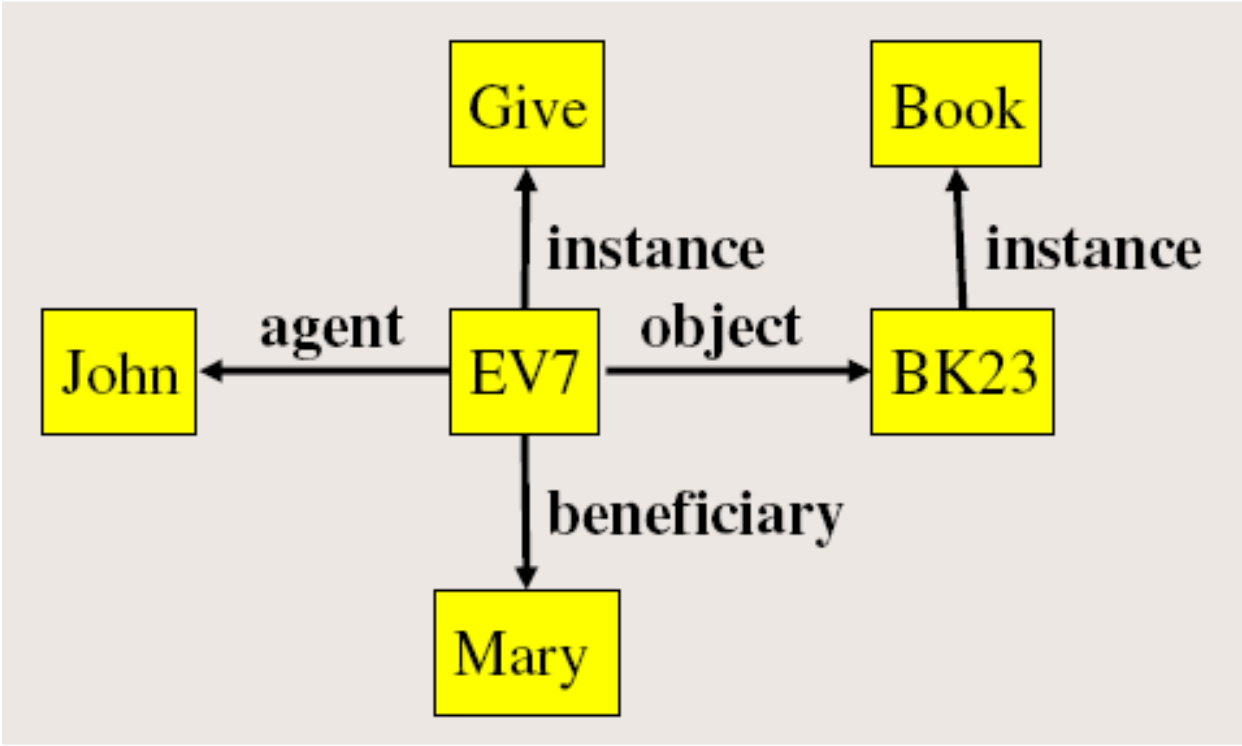
instance(Marcus,Man)

Пример 1: представяне на n-арна релация, n>2



score(Cubs, Dodgers, 5-3)

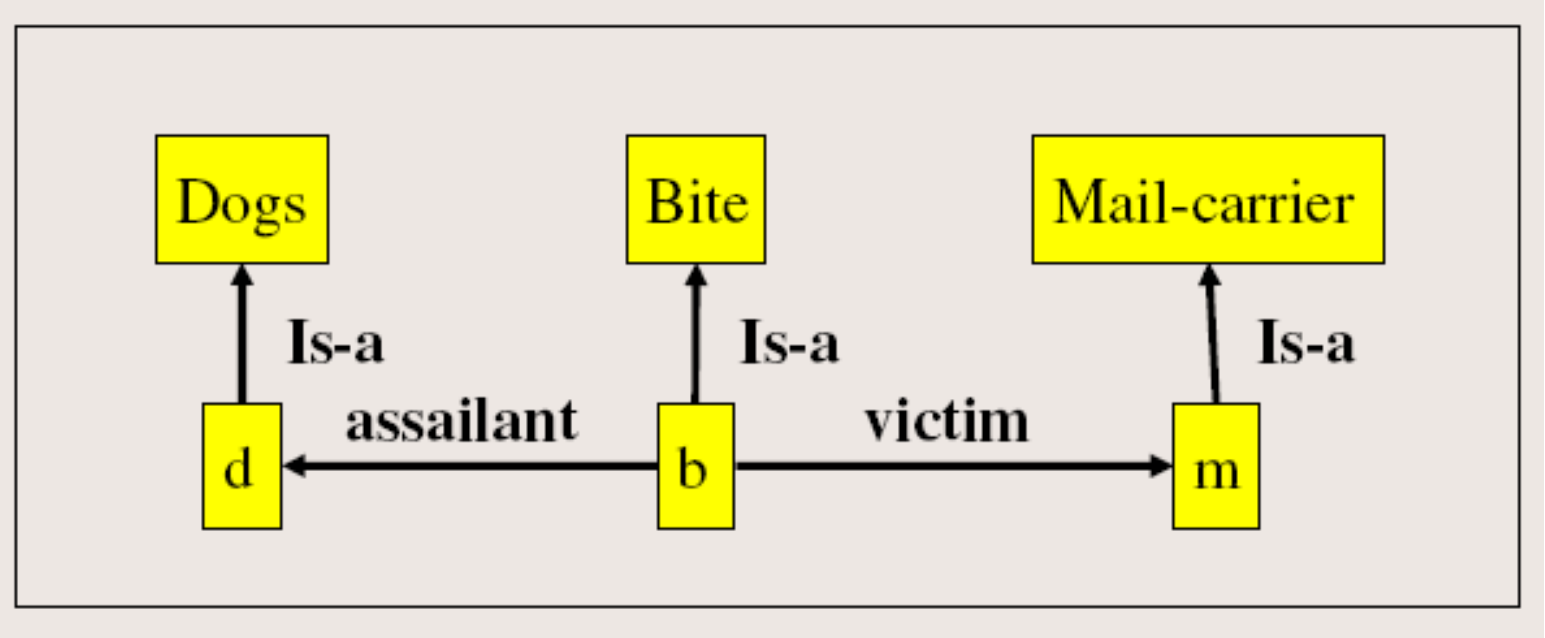
Пример 2: представяне на изречение на естествен език



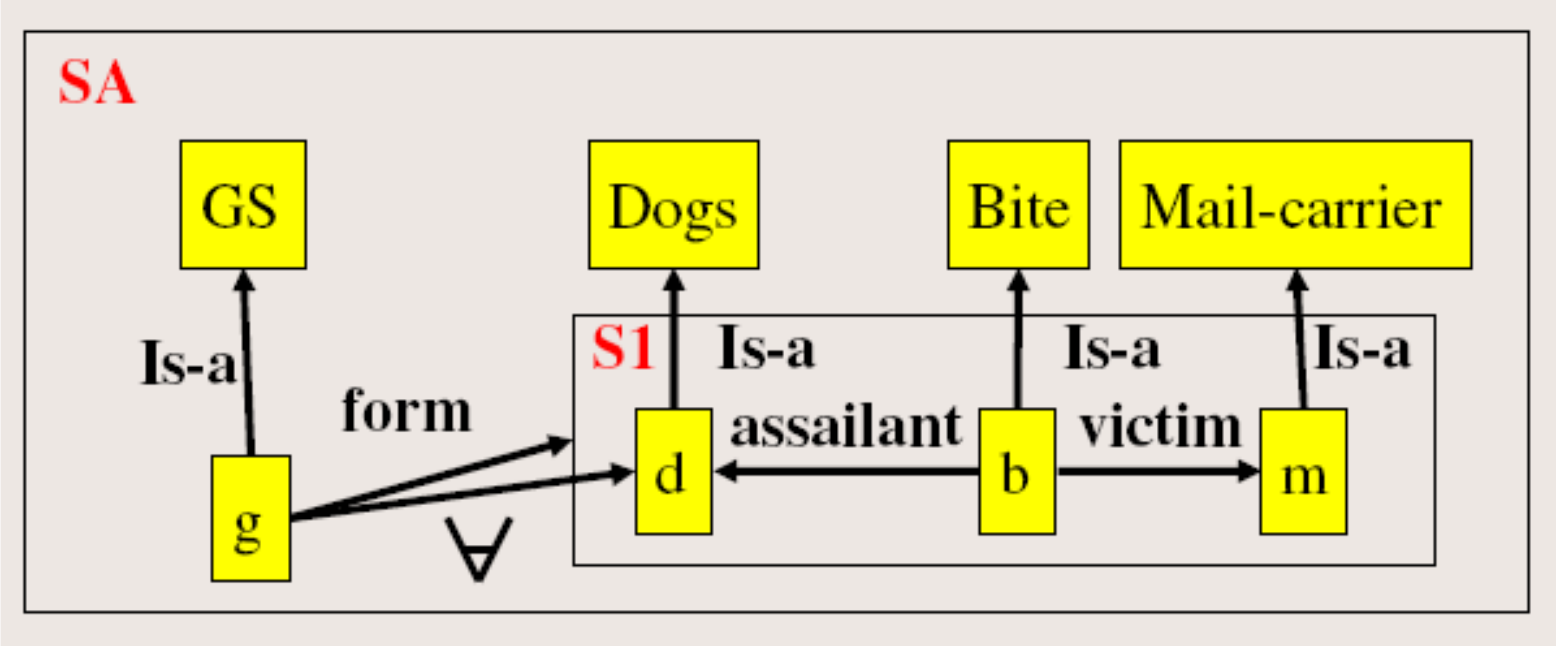
“John gave the book to Mary.”

Пример 3: използване на разделена СМ (partitioned SN)

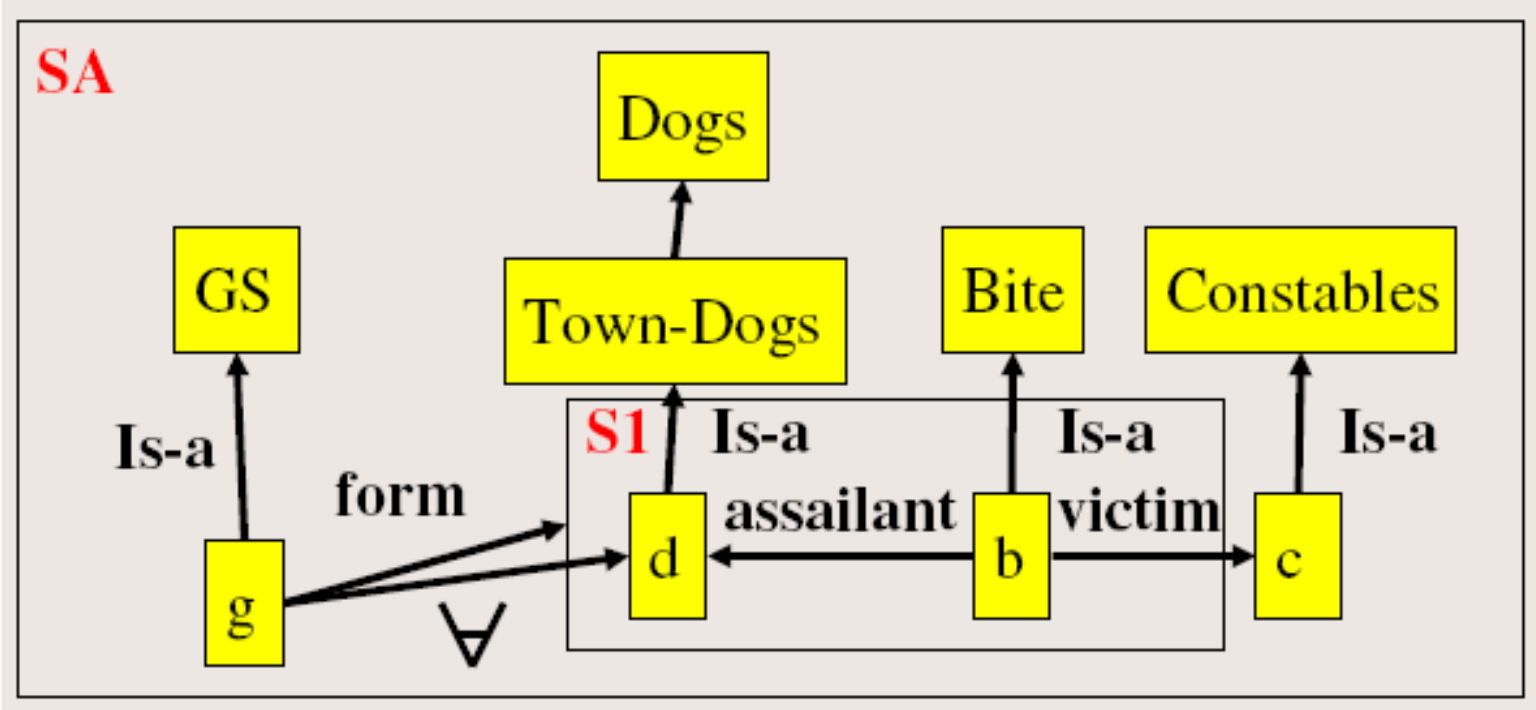
a) The dog bit the mail carrier.



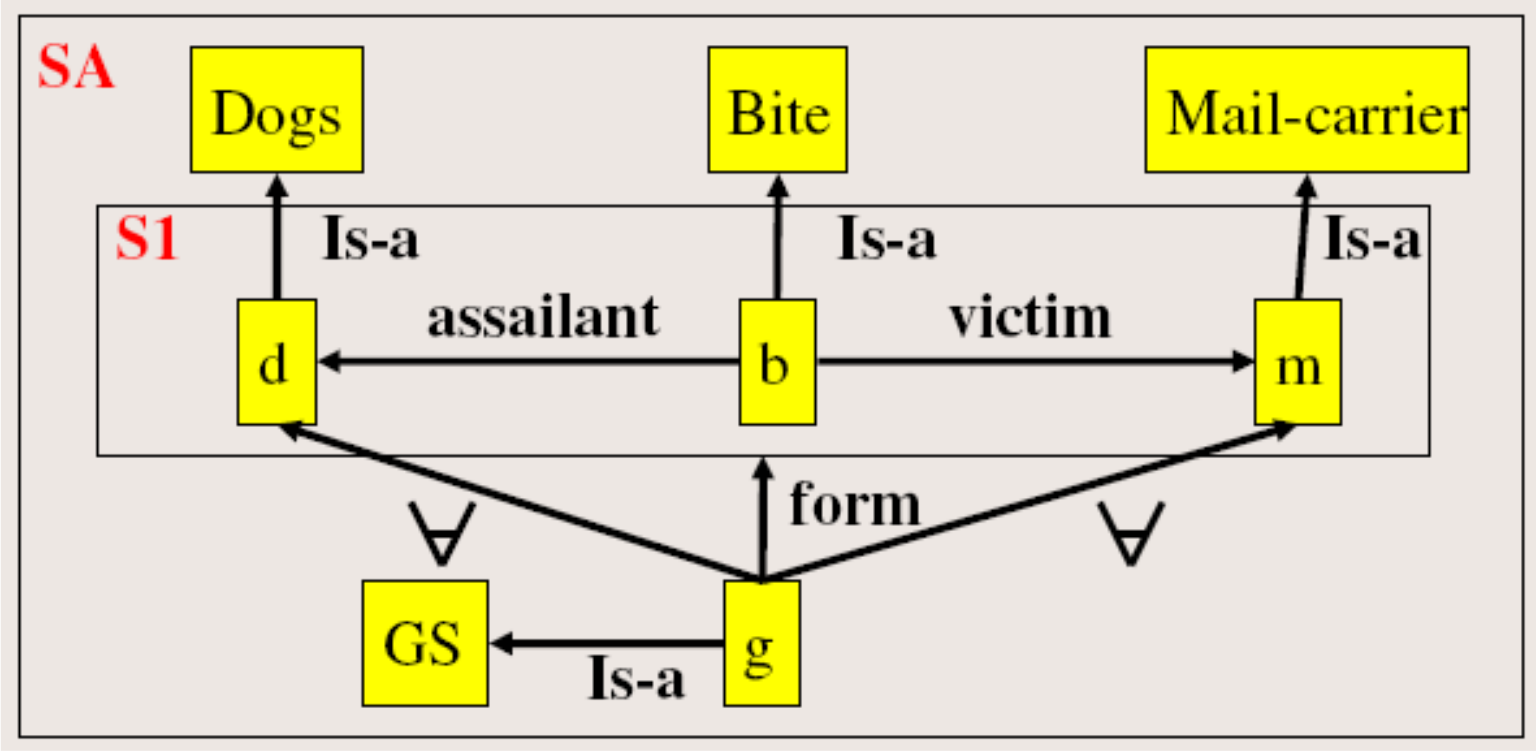
b) Every dog has bitten a mail carrier.



c) Every dog in town has bitten the constable.



d) Every dog has bitten every mail carrier.



Обща оценка на семантичните мрежи като формализъм за ПИЗ

Специфични преимущества: простота, нагледност, яснота, естественост на представянето.

Недостатъци и проблеми:

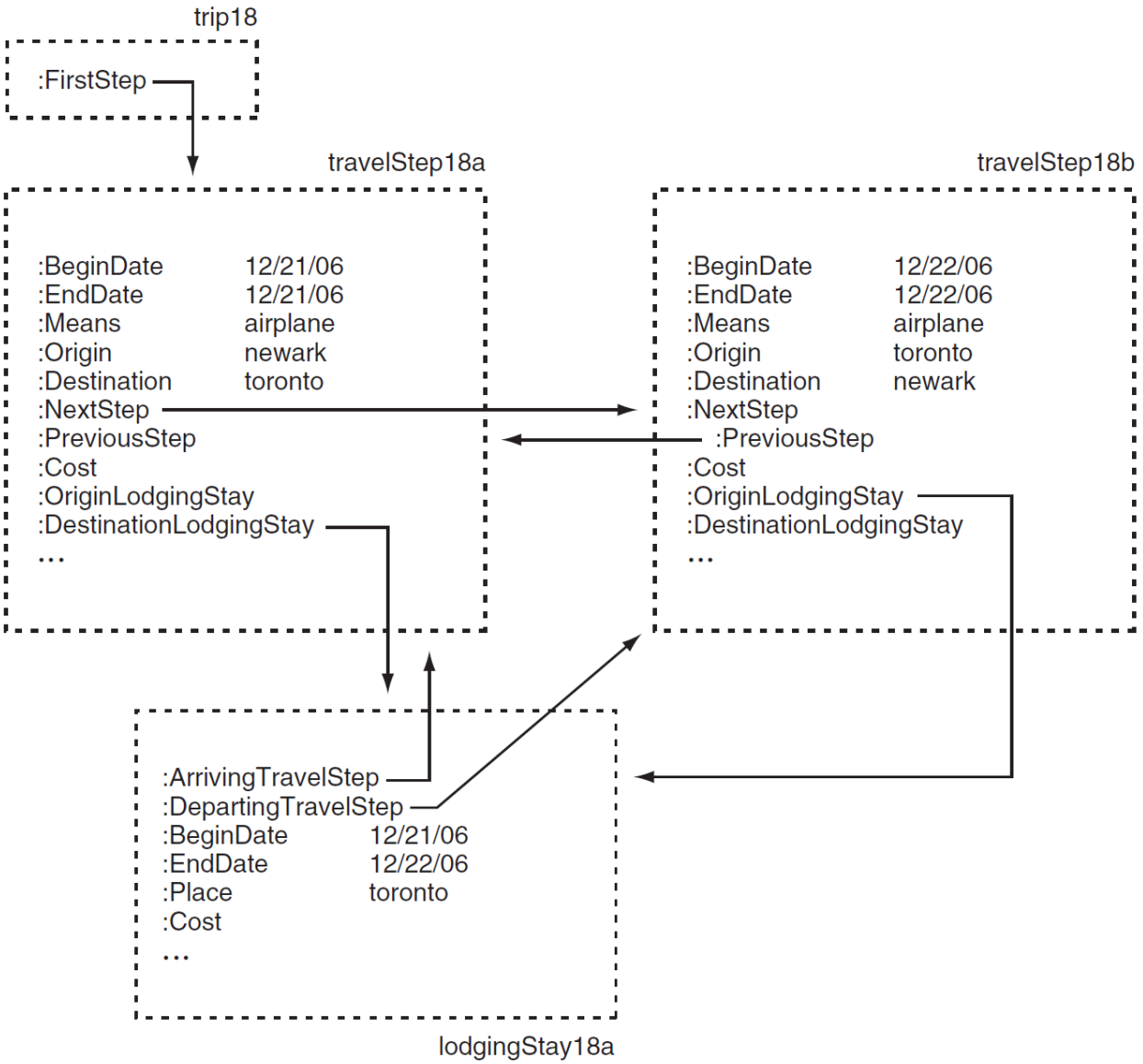
- недостатъчна изразителна сила;
- неясна семантика;
- проблеми при извършването на различни операции със семантични мрежи;
- проблеми при управлението на наследяването.

Представяне и използване на знания чрез фреймове

Обща характеристика

Фреймът е структура от данни, предназначена за описание на дадена стандартна, стереотипна ситуация (М. Мински). Идеята за фреймовете съответства на създаването и използването на представи, аналогии и натрупан опит в човешкото мислене.

Според някои автори фреймовете са структури, всяка от които се състои от име и множество двойки *атрибут – стойност*. Името отговаря на възел в семантична мрежа, атрибутите – на имената на дъгите, излизащи от този възел, а стойностите – на възлите, до които водят дъгите. Фреймът може да включва и *метазнания* (информация за самия фрейм или цялата фреймова система, а не за представяния чрез фрейма обект).


trip18
:FirstStep
travelStep18a
:BeginDate 12/21/06
:EndDate 12/21/06
:Means airplane
:Origin newark
:Destination toronto
:NextStep
:PreviousStep
:Cost
:OriginLodgingStay
:DestinationLodgingStay
...
travelStep18b
:BeginDate 12/22/06
:EndDate 12/22/06
:Means airplane
:Origin toronto
:Destination newark
:NextStep
:PreviousStep
:Cost
:OriginLodgingStay
:DestinationLodgingStay
...
:ArrivingTravelStep
:DepartingTravelStep
:BeginDate 12/21/06
:EndDate 12/22/06
:Place toronto
:Cost
...
lodgingStay18a

Атрибутите могат да имат стойности по подразбиране. Могат да се задават условия и ограничения за стойностите им или процедури, които да се извикват при добавяне или опит за достъп до стойността на даден атрибут. Обектът може да бъде представен не чрез един фрейм, а чрез фреймова система, в която на всеки от няколко възможни (разглеждани) аспекта на обекта съответства отделен фрейм, свързан с останалите чрез явно зададени връзки или общи стойности на определени атрибути.

Затова е по-точно да се каже, че фреймът е структура от данни, която включва три нива на йерархия:

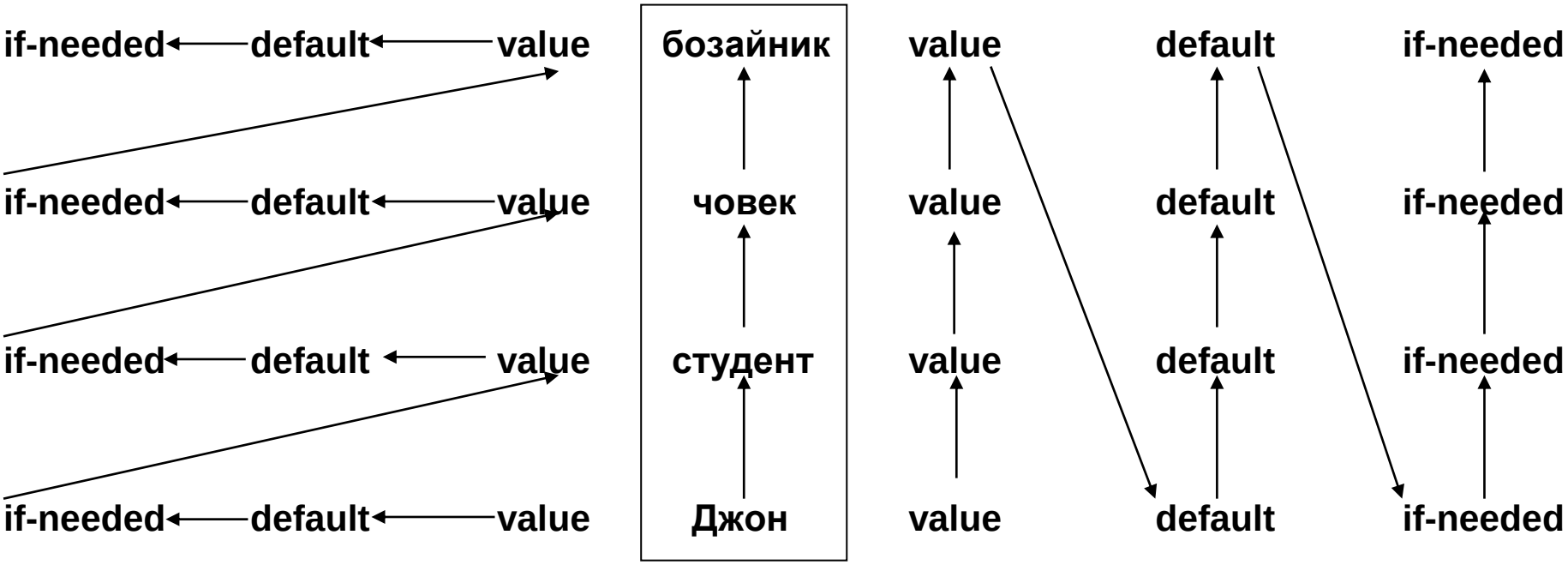
- т. нар. self информация (името на фрейма и др.);
- множество от *слотове* (атрибути);
- множество от *фасети*, асоциирани със слотовете.

Основни типове фасети в езика FRL (Frame Representation Language):

- value
- default
- if-needed
- if-added
- if-removed

Описание на родово-видови отношения (обект – клас, клас – суперклас) в езика FRL: слот A-K-O (A Kind Of).

Основни стратегии за търсене (наследяване на свойства) в езика FRL: I-търсене, Z-търсене, N-търсене.



Z-търене

N-търсене

Обща оценка на фреймовете като формализъм за ПИЗ

Специфични преимущества:

- естественост на представянето (представянето е близко до начина, по който се съхраняват знания за различните понятия в човешкия мозък);
- модулност и йерархична структура на базата от знания;
- добри възможности за задаване на свойства (стойности), които са валидни по подразбиране.

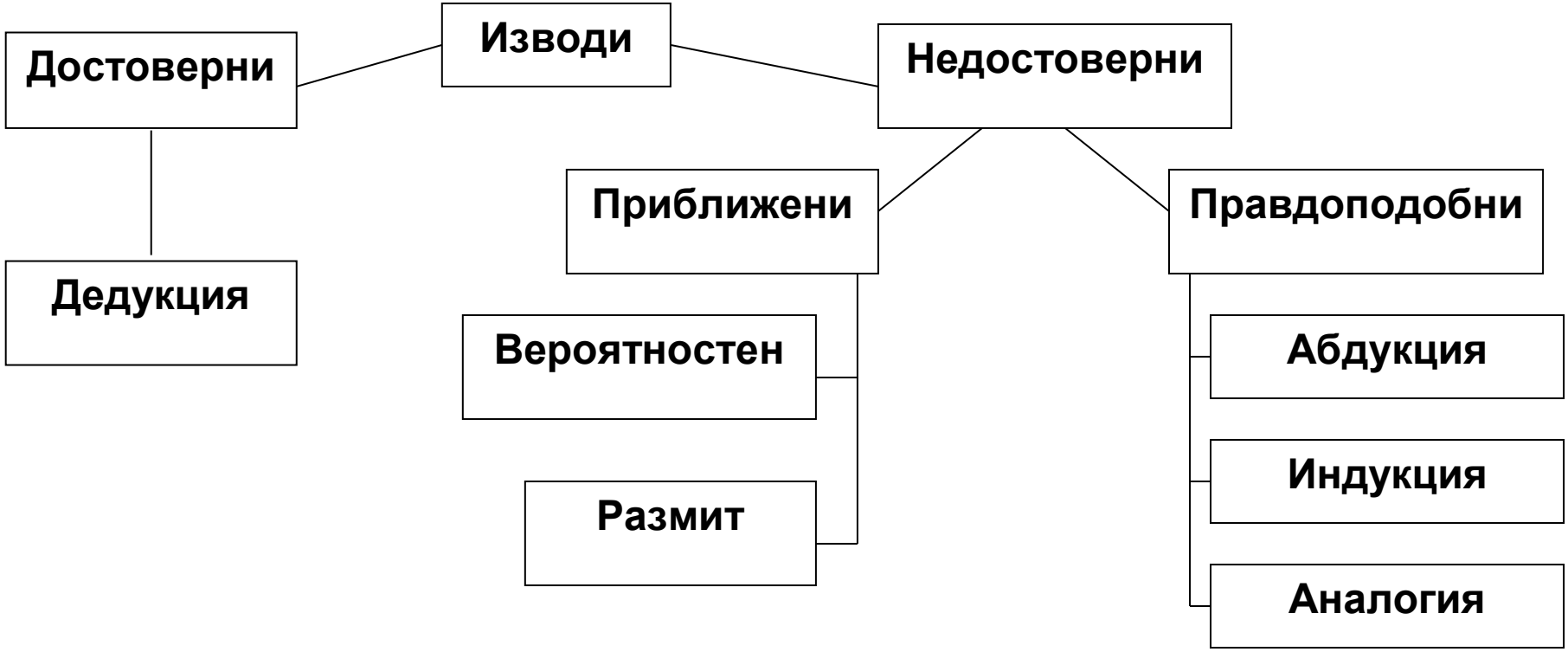
Недостатъци и проблеми:

- неясна семантика;
- недостатъчна изразителна сила;
- проблеми при управлението на наследяването.

Системи, основани на знания - зимен семестър,
2022/2023 учебна година

Тема 10:
Работа с несигурни знания в СОЗ

Видове изводи в СОЗ



Дедукция. Теоретична основа на дедукцията е правилото за извод Modus Ponens (MP). Същност на MP:

(ако A, то B)	условно вярно твърдение
A	изпълнено условие
B	изведено заключение

По-точно, дедуктивният извод има формата:

(∀x) (ако A(x), то B(x))
A(a)
B(a)

При това като модел на твърденията от вида (ако A, то B) обикновено се използва традиционната импликация A -> B. В действителност тя не означава непременно причинно-следствена връзка между A и B, но в повечето случаи върши работа (същевременно импликацията е добре формализирана, а причинността е много сложно за моделиране понятие).

Интерпретаторът на правилата в системите, основани на правила, по същество извършва дедуктивен извод (прав или обратен).

Абдукция. Абдукцията е генериране на правдоподобни обяснения за това, което наблюдаваме около нас. Тя може да се разглежда в следната форма:

(ако A, то B)
B

A

По-точно, абдукцията би трябвало да се разглежда като правило за извод от вида

(причина *?x ?y*)
?y

?x

Пример. Когато хората са пияни, те не могат да пазят равновесие. Ако Джак не може да пази равновесие, бихме могли да предположим, че той е пиян. Естествено, това е само едно предположение, което може да се окаже и невярно (причината за неспособността му да пази равновесие може да бъде съвсем друга).

Индукция. Индуктивният извод е опит за обобщение на базата на общи признаци, наблюдавани у голям брой конкретни обекти. При натрупване на допълнителни знания за средата достоверността на обобщението може съществено да се повиши. От тази гледна точка може да се твърди, че способността за индуктивен извод е съпоставима със способността на човека за обучение и самообучение.

Аналогия. При извода по аналогия на базата на знания за сходство между два обекта по някои признаци се генерира хипотезата, че тези обекти са сходни и по други признаци, които са установени в единия обект, но все още не са установени в другия. В този смисъл при извода по аналогия се извършва пренасяне (трансформиране) на информация от единия обект към другия.

Представяне на несигурни знания и вероятностни разсъждения

Представяне на несигурни знания с вероятности

- *Случайна променлива.* Величина в езика за представяне на знания, която може да има няколко (вкл. безброй много) възможни стойности.
- *Област на променлива*: *dom(x)* = множеството от възможни стойности на *x*.
- *Твърдение*: булев израз от присвоявания на променливи ($x_i = v_j$). Например: *(време = дъждовно)* $\vee$ *(болест = грип)* $\vee$ $\neg$*(температура = повишена)*.

- *Вероятност* = мярка за увереност в дадено твърдение (реално число между 0 и 1). *P(A)*=0 $\rightarrow$ 100% увереност, че твърдението *A* е лъжа; *P(A)*=1 $\rightarrow$ 100% увереност, че твърдението *A* е истина.
- *Вероятностното разпределение* задава вероятността на всяка възможна стойност на променливата. Ако $dom(x) = \{v_1, v_2, \ldots, v_n\}$, то $\sum_{i=1}^{n} P(x = v_i) = 1$.
- *Априорна вероятност* – вероятност при отсъствие на каквато и да е информация.
- *Условна вероятност* – вероятност при наличието на информация за стойностите на други случайни променливи. Например: *P(температура = повишена | болест = грип)*.

- *Основни зависимости* (*A*, *B* – твърдения):
  - $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
  - *A* и *B* са *независими* (т.е. знанието на едното не променя вероятността на другото), когато $P(A \wedge B) = P(A)P(B)$
  - *A* и *B* са *несъвместими* (т.е. никога не могат да се случат заедно), когато
    $P(A \wedge B) = 0$
  - Дефиниция на условна вероятност: $P(A|B) = \frac{P(A \wedge B)}{P(B)}$

    Следователно, $P(A \wedge B) = P(A|B)\, P(B)$.

- *Условна независимост* на A и B при дадено C: ако $P(A|B \wedge C) = P(A|C)$ и $P(B|A \wedge C) = P(B|C)$
- Формула (теорема) на Бейс: $P(A|B) = \frac{P(B \mid A)P(A)}{P(B)}$

- Вероятностен модел на предметната област:
  - *Атомарно събитие*: $(x_1 = v_1) \wedge (x_2 = v_2) \wedge \ldots \wedge (x_n = v_n)$, където x_i са случайни променливи. Описва конкретно състояние на предметната област.
  - *Съвместно разпределение*: n-мерна таблица с m_i ($i = 1,2, \ldots, n$) клетки по всяка размерност (ако x_i има m_i възможни стойности). Във всяка клетка се записва вероятността на съответното атомарно събитие. Тъй като атомарните събития са несъвместими (т.е. взаимно изключващи се) и таблицата съдържа всички атомарни събития, то сумата от стойностите на всички клетки е 1.

Механизми за извод

- Използване на съвместното разпределение
  Дадено е съвместното разпределение на няколко случайни променливи, например

	зъбобол = да	*зъбобол = не*
кариес = да	0.04	0.06
кариес = не	0.01	0.89

Тогава могат да се изчисляват вероятностите на произволни твърдения. Например (за краткост са пропуснати стойностите на променливите):

P(кариес) = 0.04 + 0.06 = 0.1 (сумата на реда)

P(кариес∨зъбобол) = 0.04 + 0.06 + 0.01 = 0.11

P(кариес|зъбобол) = P(кариес∧зъбобол)/P(зъбобол) = 0.04/(0.04+0.01) = 0.8

- Използване на формулата на Бейс
  - Дадени са: e – множество от симптоми ($e = e_1 \wedge e_2 \wedge \ldots \wedge e_k$) и $d_1, d_2, \ldots, d_n$ – изчерпващо множество от диагнози. Предполага се, че елементарните симптоми $\{e_i\}$ са независими. Известни са $P(d_i)$ и $P(e|d_i)$ за $i = 1, \ldots, n$ (по-точно, $P(e_j|d_i)$ за $j = 1, \ldots, k$ и $i = 1, \ldots, n$).
  - Задачата е да се пресметнат $P(d_i|e)$, $i = 1, \ldots, n$ и да се намери най-вероятната диагноза при дадените симптоми e.
  - Според формулата на Бейс
    $P(d_i \mid e) = \frac{P(d_i)P(e|d_i)}{P(e)}$ за всяко $i = 1, \ldots, n$

- Предполага се, че елементарните симптоми $\{e_i\}$ са независими, следователно

  $P(e \mid d_i) = \prod_{j=1}^{k} P(e_j \mid d_i)$ за всяко $i = 1, \ldots, n$

- $P(e)$ може да се намери по следния начин:

  $\sum_{i=1}^{n} P(d_i \mid e) = \sum_{i=1}^{n} \frac{P(d_i)P(e|d_i)}{P(e)} = 1$, следователно

  $P(e) = \sum_{i=1}^{n} P(d_i)P(e \mid d_i)$

- Пример:

вероятност	*здрав*	*грип*	*алергия*
P(d)	0.9	0.05	0.05
P(кихане|d)	0.1	0.9	0.9
P(кашлица|d)	0.1	0.8	0.7
P(температура|d)	0.01	0.7	0.4

Нека симптомите e са кихане и кашлица без повишена температура.

Тогава

$e = e_1 \wedge e_2 \wedge e_3$,

e_1 = *кихане*, e_2 = *кашлица*, $e_3 = \neg$(*повишена температура*)

d_1 = *здрав*, d_2 = *грип*, d_3 = *алергия*

$$P(\textit{здрав}|e) = \frac{P(\textit{здрав})P(e \mid \textit{здрав})}{P(e)} = \frac{(0.9)P(e \mid \textit{здрав})}{P(e)};$$

$$P(e|\textit{здрав}) = \prod_{j=1}^{3} P(e_j \mid \textit{здрав}) = (0.1)(0.1)(1\text{-}0.01)$$

Следовательно,

$$P(\textit{здрав}|e) = \frac{(0.9)(0.1)(0.1)(0.99)}{P(e)} = \frac{0.0089}{P(e)}$$

$$P(\textit{грип}|e) = \frac{(0.05)(0.9)(0.8)(0.3)}{P(e)} = \frac{0.01}{P(e)}$$

$$P(\textit{алергия}|e) = \frac{(0.05)(0.9)(0.7)(0.6)}{P(e)} = \frac{0.019}{P(e)}$$

$$P(e) = \sum_{i=1}^{3} P(d_i)P(e \mid d_i) = 0.0089 + 0.01 + 0.019 = 0.0379$$

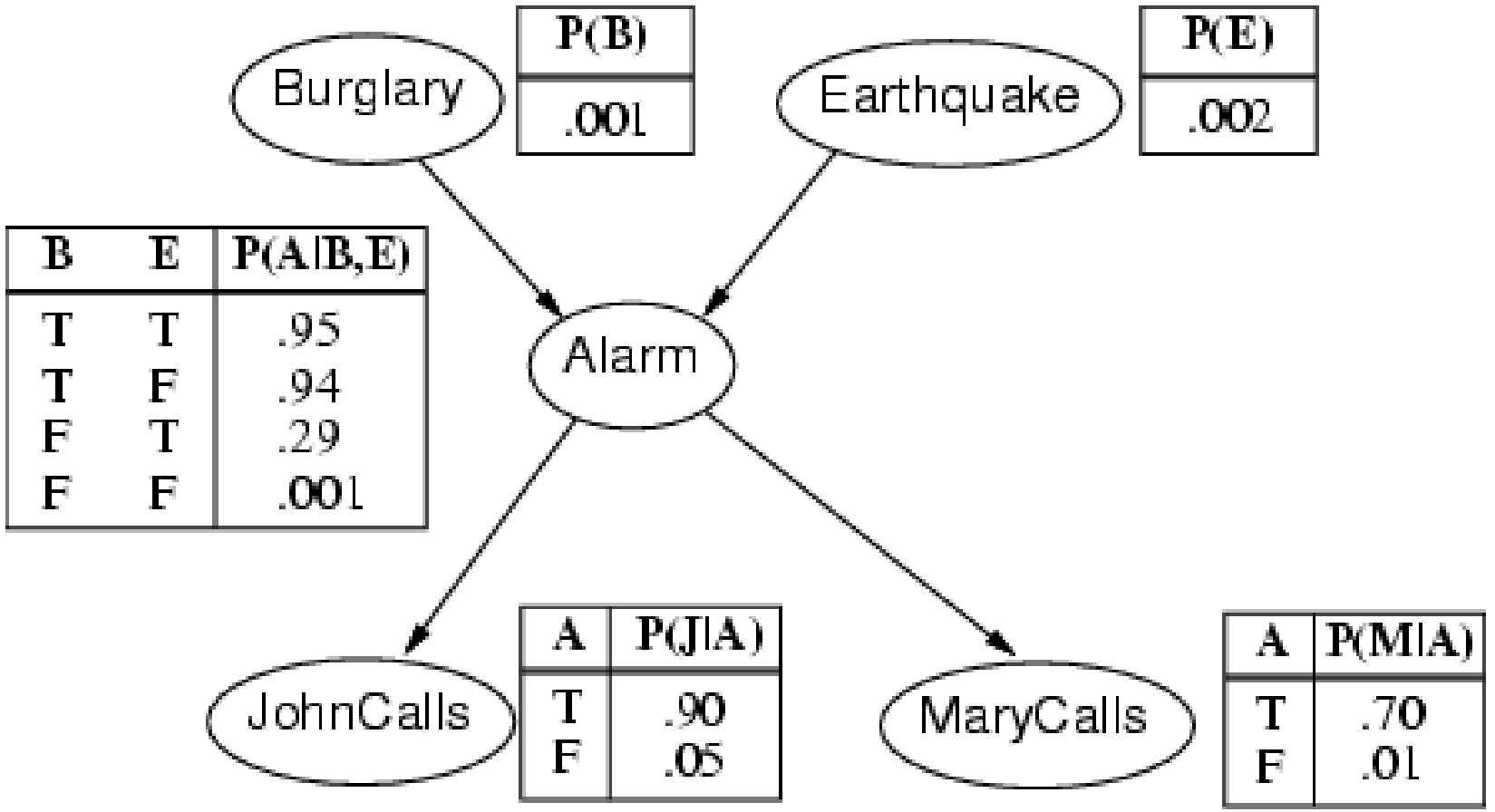
Следователно,
$P(здрав|e) = 0.23$; $P(грип|e) = 0.26$; $P(алергия|e) = 0.50$

- Проблем: предположението за независимост на елементарните симптоми е прекалено силно и нереалистично.

Бейсови мрежи (БМ; Belief Networks)

- Използване на ацикличен ориентиран граф за представяне на зависимостите между променливите с цел сбито (компактно) описание на съвместното им разпределение.
- На всяка случайна променлива съответства отделен възел от мрежата. Дъгите от мрежата задават *причинно-следствени връзки*. Интуитивното значение на дъгата от възела X към възела Y е, че X оказва *директно влияние* върху Y.
- За всеки възел е дефинирана таблица с условни вероятности, която задава вероятността на всяка стойност на променливата във възела в зависимост от всяка възможна комбинация от стойности на променливите в родителските възли.

Пример:


Burglary
P(B)
.001
Earthquake
P(E)
.002
B E P(A|B,E)
T T .95
T F .94
F T .29
F F .001
Alarm
JohnCalls
A P(J|A)
T .90
F .05
MaryCalls
A P(M|A)
T .70
F .01

Примерна предметна област. В жилището си Джейк има монтирана нова сигнална инсталация (аларма). Тя е чувствителна и реагира на опит за проникване в жилището (в частност, при опит за обир), но също и на (дори слаби) земетресения. Има също двама съседи, Джон и Мери, които са обещали да му се обаждат по телефона в офиса винаги когато чуят, че алармата в неговото жилище се е включила. Джон винаги се обажда, когато чуе алармата, но понякога я обърква със звъна на телефона и тогава също се обажда. Мери пък обича да слуша силна музика и понякога е възможно да не чуе алармата в дома на Джейк.

Ако е известно кой от двамата се е обадил или не се е обадил на Джейк, може да се установи например вероятността в жилището му да е извършен обир.

- БМ задават неявно съвместното разпределение на променливите си. Нека $x_1, x_2, \ldots, x_n$ са случайни променливи и $P(v_1, v_2, \ldots, v_n)$ е съвместната вероятност те да получат съответно стойности $v_1, v_2, \ldots, v_n$. Тогава

$$P(v_1, v_2, ..., v_n) = \prod_{i=1}^{n} P(v_i \mid Parents(x_i)),$$

  където $P(v_i \mid Parents(x_i))$ е условната вероятност за $x_i = v_i$ при условие, че са дадени стойностите на родителските променливи $Parents(x_i)$ на x_i.

Например:

$P(JohnCalls, MaryCalls, Alarm, \neg Burglary, \neg Earthqake) =$

$P(J|A).P(M|A).P(A|\neg B \wedge \neg E).P(\neg B).P(\neg E) =$

$(0.9)(0.7)(0.001)(0.999)(0.998) = 0.000628$

- Видове извод в БМ. При дадени стойности на подмножество от променливите (наблюдаеми променливи, evidence variables) да се определи вероятността на стойностите на друго подмножество от променливите (търсени променливи, query variables).
  - Диагностика – от следствието към причината:
    $P(Burglary \mid JohnCalls) = ?$
  - Предсказване – от причината към следствието:
    $P(JohnCalls \mid Burglary) = ?$
  - Междупричинен извод – между причините за дадено следствие:
    $P(Burglary \mid Earthquake) = ?$
  - Смесен извод – комбинация на горните три:
    $P(Alarm \mid JohnCalls \wedge \neg Earthquake) = ?$

Системи, основани на знания - зимен семестър,
2022/2023 учебна година

Тема 11:
Семантичен уеб и онтологии

Семантичен уеб

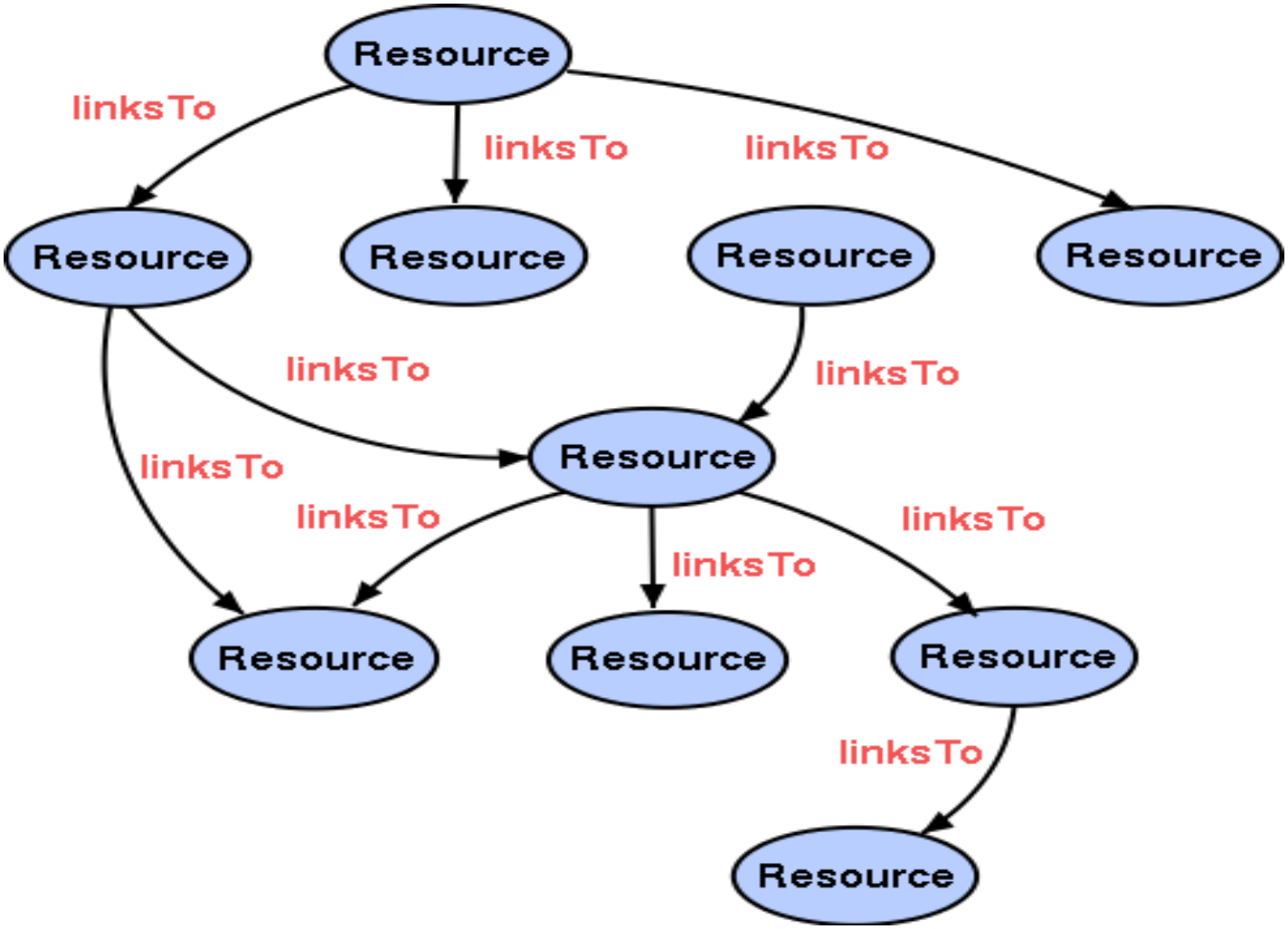
Семантичният уеб може да бъде разглеждан като разширение на традиционния уеб, в което информацията получава добре дефинирано значение, позволяващо на хората и компютрите да работят при по-добро сътрудничество.

Семантичният уеб е разширение на традиционния уеб, което позволява по-лесно да се намира, споделя и комбинира информация.

Традиционният уеб представя информация, използвайки:

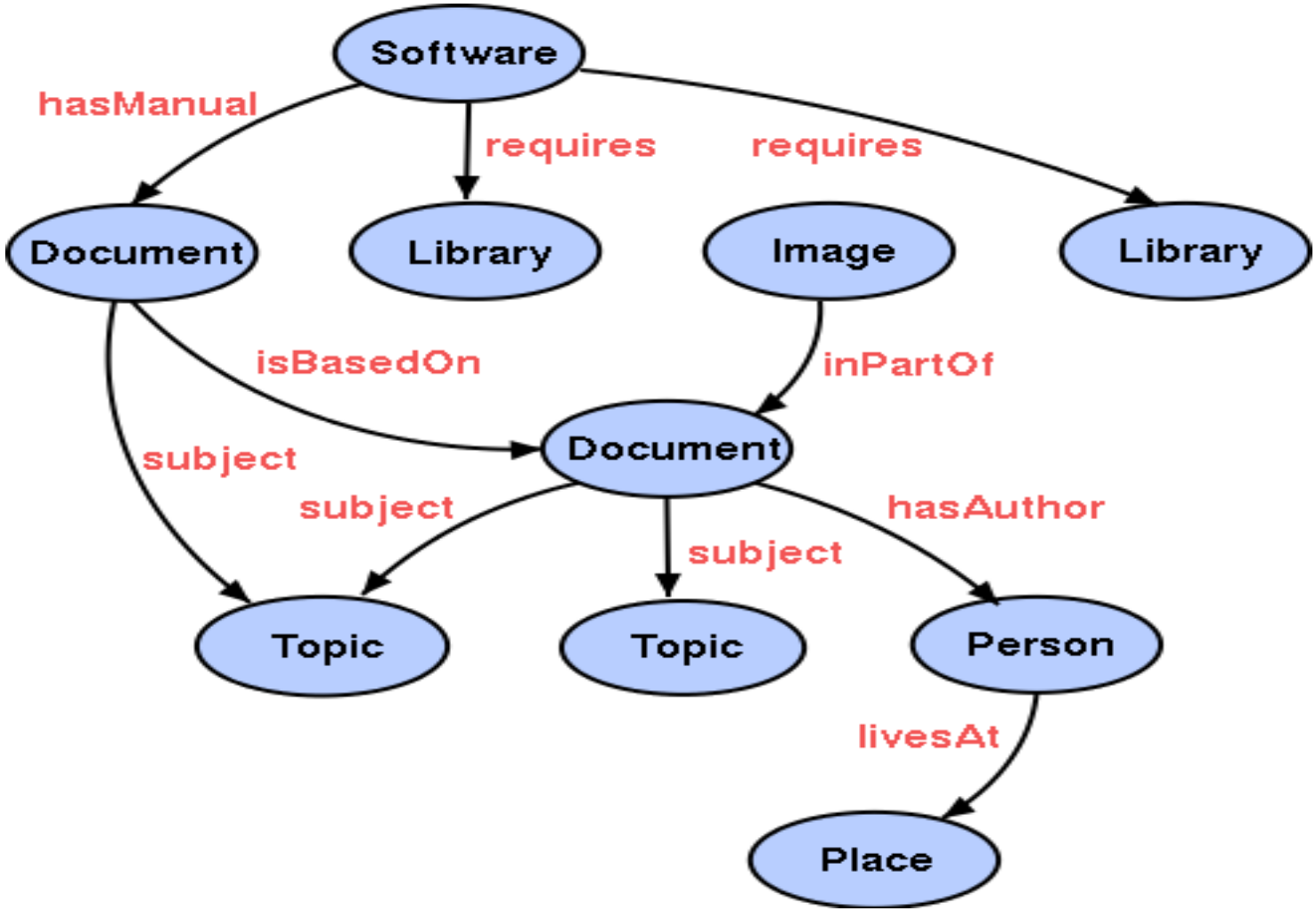
- ✓ естествен език (напр. английски)
- ✓ графика, мултимедия
- ✓ подходящо оформяне на страниците

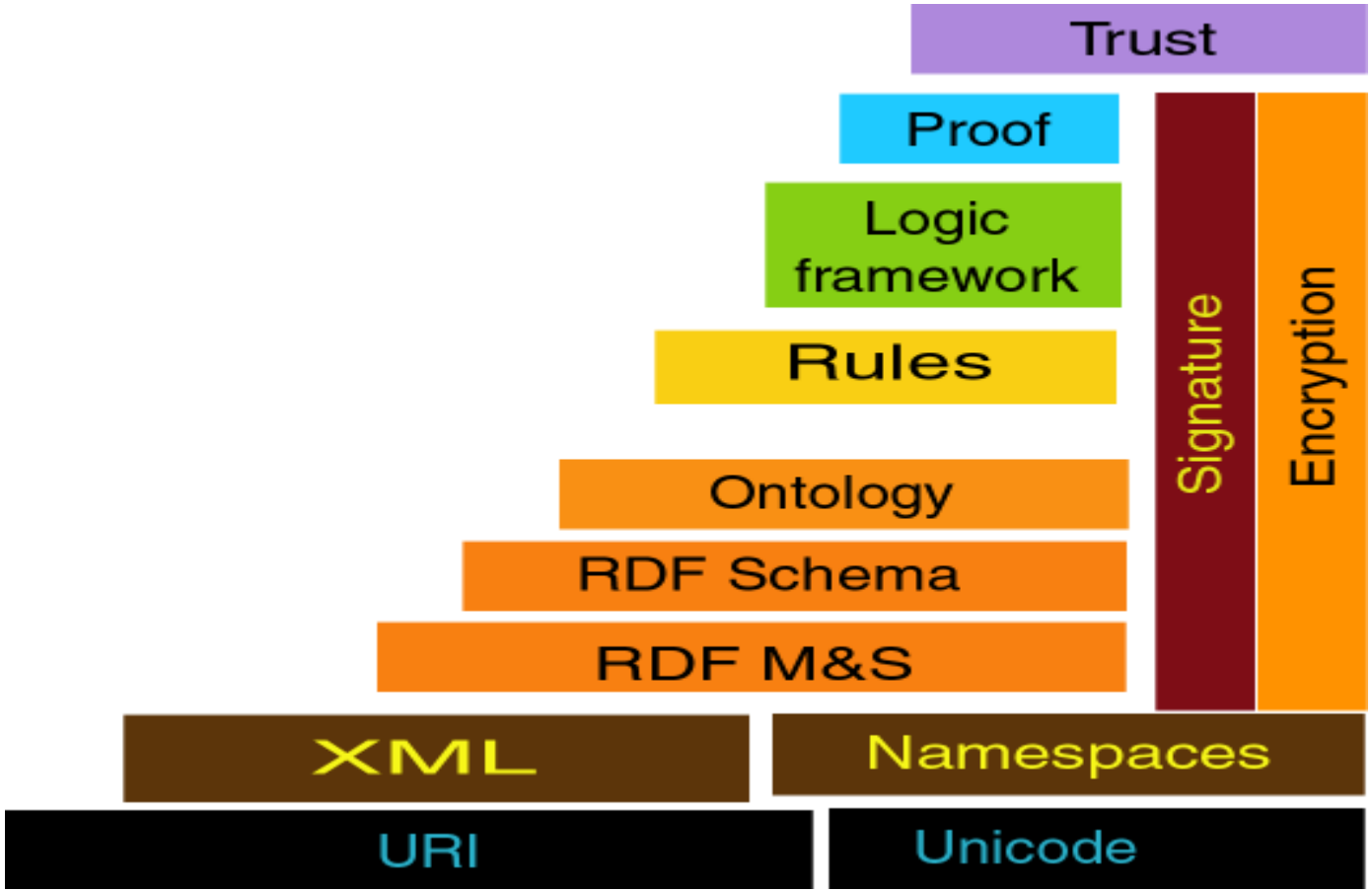
Той е труден за компютърна обработка (проблеми: многозначност, неподходящи формати на данните, неинформативност и нерационалност на връзките между отделните информационни ресурси).


Resource
linksTo
linksTo
linksTo
Resource
Resource
Resource
Resource
linksTo
linksTo
linksTo
Resource
linksTo
linksTo
linksTo
Resource
Resource
Resource
linksTo
Resource

Семантичният уеб:

- ✓Разширява традиционния уеб
- ✓Позволява информацията да бъде представяна във формат, който предполага еднозначна интерпретация и е удобен за компютърна обработка
- ✓Позволява да бъдат добавяни подходящи метаданни за описание на съществуващи или новопостъпващи данни


Software
hasManual
requires
requires
Document
Library
Image
Library
isBasedOn
inPartOf
Document
subject
subject
subject
hasAuthor
Topic
Topic
Person
livesAt
Place



Технологии на семантичния уеб

Онтологии

Онтология (от древногръцки: онтос – битие, съществуване; логос – учение, наука) е термин, определящ науката за битието, за съществуващото, в отличие от ***гносеологията*** – науката за познанието. Терминът “онтология” в съвременната философска литература се използва за означаване на определена система от категории, които са следствие от определена система от възгледи (определена гледна точка) за света.

В литературата по изкуствен интелект “онтология” е термин, който се използва за означаване на формално представени знания на основата на някаква ***концептуализация***. Концептуализацията предполага описание на множество от обекти и понятия, знания за тях и връзки между тях.

Според Т. Грубер ***онтология се нарича експлицитната спецификация на концептуализацията***. Формално онтологията се състои от термини, организирани в таксономия, техните определения и атрибути, а също и свързаните с тях аксиоми и правила за извод.

С други думи, онтологията е ***база от знания, описваща факти, за които се предполага, че са винаги верни в рамките на определена взаимна общност*** на основата на общоприетия смисъл на използвания речник.

В специализираната литература напоследък се налага следното определение: ***онтологиите са БЗ от специален тип, които могат да се „четат“ и разбират, да се отделят от разработчика и/или физически да се поделят между техните потребители***.

Свойства на онтологиите

Задължителни свойства:

- Наличие на краен разширяем речник
- Възможност за еднозначна интерпретация на класовете и релациите
- Йерархична структура на системата от класове

Типични свойства:

- Възможност за спецификация на свойства на отделните класове
- Възможност за създаване на *индивиди* (екземпляри на класовете)
- Възможност за спецификация на ограничения върху стойностите на свойствата

Препоръчителни свойства:

- Възможност за спецификация на непресичащи се класове
- Възможност за спецификация на произволни релации между термове
- Възможност за спецификация на определени видове релации (свойства) като *обратно* свойство (inverse property, например *parent↔child*), *симетрично* свойство или релация *part-whole*

Възможни приложения на онтологиите

- Предоставят речник на предметната област, който може да бъде използван като основа на общуването между автори, потребители и програмни системи.
- Могат да се използват при проектирането на структурата на уеб сайтове и нивата на достъп до тях.
- Могат да се използват за бърза проверка дали даден уеб сайт отговаря на очакванията на съответния потребител.

- Могат да се използват за подходящо разширяване на потребителските заявки за търсене в Интернет.
- Могат да се използват за ограничаване на търсенето в Интернет чрез премахване на опасността от многозначна интерпретация на съответната потребителска заявка.
- Могат да се използват при проектиране на интелигентни потребителски интерфейси на системи за прогнозиране, диагностика и др.

- Използват се като източници на знания при семантично анотиране и семантично обогатяване (semantic enrichment) на данни или различни типове съдържание (текстове, изображения, видео и др.).
- Използват се като източници на знания при интегрирането на данни и осигуряването на семантична оперативна съвместимост (semantic interoperability) на хеторогенни информационни системи.

Системи, основани на знания - зимен семестър, 2022/2023 учебна година

Тема 12:

Средства за описание на информационни ресурси в RDF/RDFS

RDF

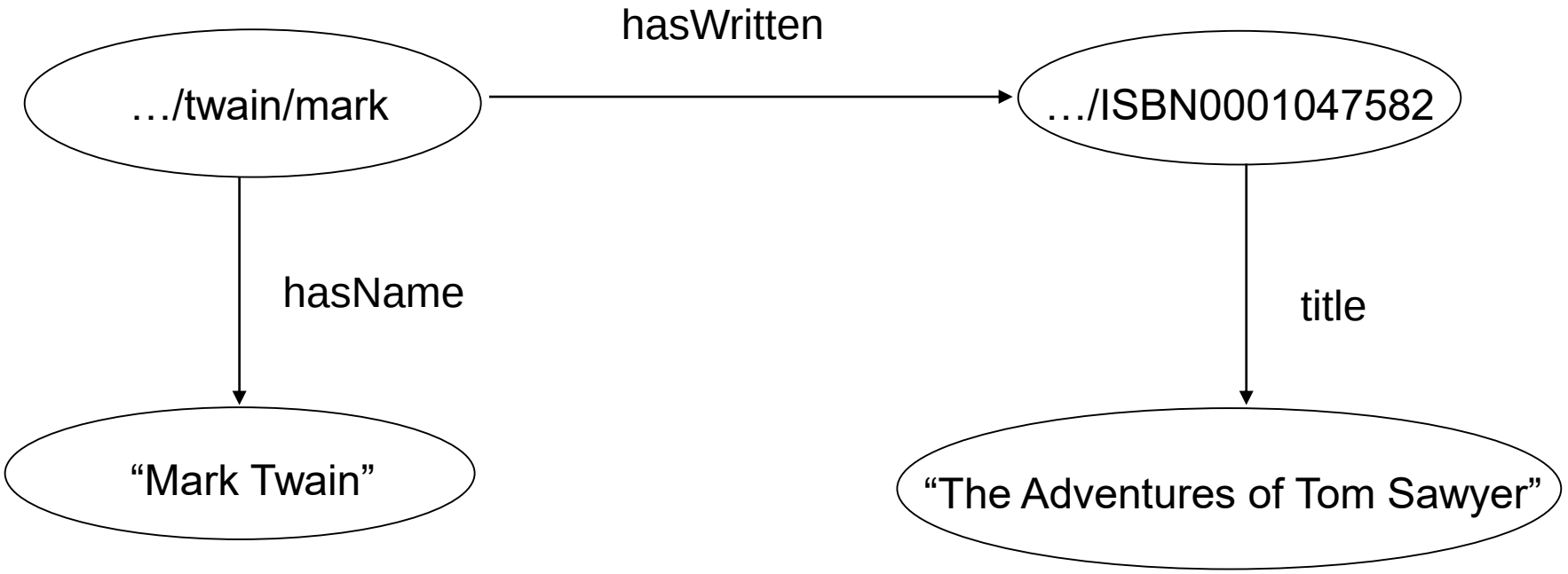
RDF (Resource Description Framework) е препоръчителна рамка на W3C, проектирана с цел стандартизиране на дефинирането и използването на описания на метаданни за уеб-базирани ресурси. Същевременно, RDF е подходящо средство за представяне на произволни данни (а не само на метаданни).

Основни конструкции в RDF са *тройките* обект-атрибут-стойност (object-attribute-value). Такава тройка обикновено се записва като A(O;V) и означава, че обектът O има атрибут A със стойност V.

Друг популярен подход е основан на разглеждането на релация от посочения тип като именувана (етикетирана) дъга между два възела в ориентиран граф: *[O]_A![V].*

Тази нотация е полезна, тъй като RDF позволява обектите и стойностите да бъдат взаимно заменяни. Всеки обект от една тройка може да играе ролята на стойност от друга тройка и обратно и това може да бъде изразено лесно в рамките на дискутираното графично представяне.

Примерен RDF граф, обединяващ три твърдения:



Графът от предходната фигура представя следната система от релации:

hasName(‘http://www.famouswriters.org/twain/mark/’,“Mark Twain”)

hasWritten(‘http://www.famouswriters.org/twain/mark/’,‘http://www.books.org/ISBN0001047582’)

title(‘http://www.books.org/ISBN0001047582’,“The Adventures of Tom Sawyer”)

Възможно е също така да бъде означено, че даден обект е от определен тип, например да се специфицира, че “ISBN0001047582” е от тип Book, чрез създаване на специална дъга, която сочи към дефиницията на типа Book в съответната RDF схема:

```
type('http://www.books.org/ISBN0001047582','http://www.description.org/schema#Book')
```

Моделът и синтактичната спецификация на RDF предполагат XML синтаксис на моделите на данни в RDF. Едно възможно описание на разгледаните по-горе релации би изглеждало примерно по следния начин:

```
  <rdf:Description
rdf:about="http://www.famouswriters.org/twain/mark">
  <s:hasName>Mark Twain</s:hasName>
  <s:hasWritten
rdf:resource="http://www.books.org/ISBN0001047582"/>
  </rdf:Description>
  <rdf:Description
rdf:about="http://www.books.org/ISBN0001047582">
  <s:title>The Adventures of Tom Sawyer</s:title>
  <rdf:type
rdf:resource="http://www.description.org/schema#Book"/>
  </rdf:Description>
```

Използваният тук синтаксис е само една от многото възможности за описание на RDF модел със средствата на XML.

Важно е да се отбележи, че RDF е проектиран с цел да предложи базов модел (базиран на тройки от вида обект-атрибут-стойност) за уеб-достъпни данни.

Съществено ограничение на модела на данни, поддържан от RDF, е липсата на каквито и да е съглашения по отношение на имената и семантиката на използваните термове (релации).

RDF Schema

RDF Schema е формализъм, който предоставя на разработчиците възможности да дефинират специфични речници на атрибутите на RDF данните и да определят типовете обекти, към които могат да бъдат „прилагани“ тези атрибути.

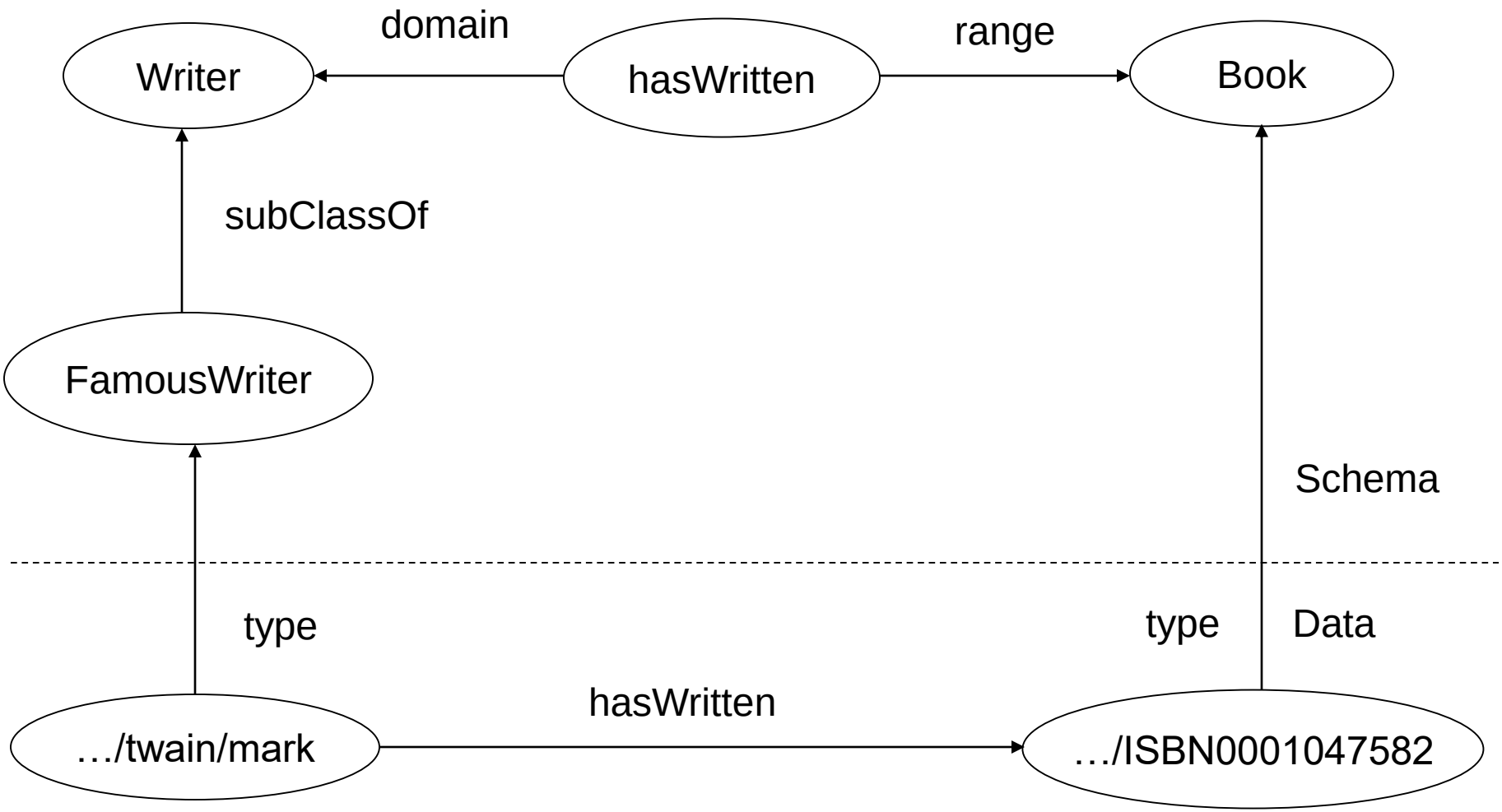
За целта RDF Schema поддържа добре дефинирана терминология, която включва в частност понятия като Class, subClassOf, Property и др., които могат да бъдат използвани в конкретни за съответните приложения схеми.

По-точно, изразите в RDF Schema са „нормални“ RDF изрази, в които е валидно предварително споразумение относно *семантиката* на определени термове и, следователно, относно *интерпретацията* на определени твърдения.

Например, свойството subClassOf позволява на разработчика да специфицира подходяща *йерархична организация на класовете*. Възможно е да се дефинират обекти като *екземпляри* на класове, използвайки свойството type. С използване на конструкциите domain и range може да се специфицират *ограничения* върху употребата на свойствата.

Над пунктираната линия на следващата фигура е изобразена примерна RDF схема, която дефинира речника за дискутирания по-рано RDF пример: Book, Writer и FamousWriter са специфицирани като класове, а hasWritten се определя като свойство. Конкретен екземпляр е описан в термините на този речник под пунктираната линия на фигурата.

Примерна RDF схема, дефинираща речник и йерархия на класове:



Резюме

RDF документите и RDF схемите могат да бъдат разглеждани на три различни нива на абстракция:

1. На *синтактично ниво* те са XML документи.

2. На *структурно ниво* те представляват множества от тройки (triples).
3. На *семантично ниво* те се състоят от множества от графи с частично дефинирана семантика.

Системи, основани на знания - зимен семестър, 2022/2023 учебна година

Тема 13: Въведение в езика OWL

Езикът OWL (Web Ontology Language) е създаден с цел да може да бъде използван за автоматична (компютърна, програмна) обработка на съдържанието на информационни ресурси, а не само за представяне на информация, предназначена за използване от хора.

В този смисъл OWL предоставя допълнителни (в сравнение с RDF/RDF Schema) възможности за „внасяне“ на семантика (значение) в описанията на съответните информационни ресурси.

OWL разширява (в сравнение с RDFS) речника, който може да бъде използван за описание на свойствата и класовете. В частност, възможно е да се описват релации между класове (например непресичащи се класове), кардиналност (например „точно един ...“), характеристики на свойствата (например симетричност) и др.

Подмножества на езика OWL

OWL включва три подмножества (езикови нива): OWL Lite, OWL DL и OWL Full. Тези подмножества са с нарастваща изразителна сила (в посочения ред) и са предназначени за използване от различни типове разработчици и крайни потребители.

- *OWL Lite* е полезен за тези потребители, които се нуждаят от средство за описание предимно на таксономии (класификационни йерархии) и прости ограничения.

- *OWL DL* е полезен за тези потребители, които имат потребност от максимална изразителна сила при запазване на изчислителната пълнота (за всички възможни заключения е гарантирано, че са изчислими).

OWL DL включва всички езикови конструкции, но те могат да бъдат използвани при определени ограничения (например, един клас може да бъде подклас на много класове, но не е допустимо един клас да бъде екземпляр на друг клас). Формална онова на OWL DL са т. нар. *description logics*.

- *OWL Full* е предназначен за потребители, които предпочитат максимална изразителна сила и синтактична свобода, без гаранции за изчислимост.

Кратък преглед на конструкциите на OWL Lite

OWL Lite поддържа част от възможностите на езика OWL и налага ограничения върху използването на някои езикови конструкции. OWL Lite може да се разглежда като подмножество на OWL DL, съответно OWL DL представлява подмножество на OWL Full.

Конструкции на OWL Lite, „наследени“ от RDF/RDFS

- ***rdfs:Class***: Всеки клас представлява множество от индивиди (индивидуални обекти), които са обединени от обстоятелството, че имат общи свойства. Класовете могат да бъдат организирани в йерархия с помощта на конструкцията *subClassOf*.

Съществува един максимално общ вграден клас, наречен Thing, който е суперклас на всички OWL класове. Съществува също един вграден клас, наречен Nothing, който няма екземпляри и е подклас на всички OWL класове.

- ***rdfs:subClassOf***: Позволява дефиниране на йерархии от класове, като специфицира даден клас като подклас на друг. Например, класът Person би могъл да бъде специфициран като подклас на класа Mammal.

По такъв начин всяка програма за логически извод (всеки ***reasoner***) ще може да направи заключение, че ако един обект е екземпляр на класа Person, то той същевременно е екземпляр на класа Mammal.

- ***rdf:Property***: Свойствата (properties) дефинират релации между индивидуални обекти или релации между индивидуални обекти и данни от определени типове. Примери за свойства: hasChild, hasRelative, hasSibling, hasAge. Класовете owl:ObjectProperty и owl:DatatypeProperty са подкласове на RDF класа rdf:Property.

- ***rdfs:subPropertyOf***: Позволява дефиниране на йерархии от свойства. Например, hasSibling може да се специфицира като подсвойство (subproperty) на hasRelative.

По този начин, ако един обект е свързан с друг чрез свойството hasSibling, то тези два обекта са свързани и чрез свойството hasRelative и това твърдение трябва да може да бъде изведено от всяка програма за логически извод.

- ***rdfs:domain***: Дефиниционната област (областта; domain) на едно свойство определя/ограничава множеството от индивиди, към които може да бъде прилагано това свойство.

- ***rdfs:range***: Определя/ограничава множеството от индивиди, които могат да бъдат стойност на съответното свойство.

- ***Individual***: Индивидите (индивидуалните обекти) се дефинират като екземпляри на класове и могат да бъдат свързвани чрез свойства.

Равенство и неравенство в OWL Lite

- ***equivalentClass***: Позволява да се специфицира, че два класа са еквивалентни. Еквивалентните класове имат едни и същи екземпляри. Равенството може да се използва за създаване на класове – синоними.

- ***equivalentProperty***: Позволява да се специфицира, че две свойства са еквивалентни. Равенството може да се използва за създаване на свойства – синоними.

- ***sameAs***: Позволява да се специфицира, че два индивидуални обекта са еквивалентни. По такъв начин могат да бъдат създавани обекти, които имат много имена.

- ***differentFrom***: Позволява да се специфицира, че даден индивид е различен от други индивиди. Възможността явно да се посочи, че определени индивиди са различни, е съществена, тъй като езиците от типа на OWL (и RDF) не предполагат, че индивидите имат само по едно име.

Характеристики на свойствата в OWL Lite

- ***inverseOf***: Позволява едно свойство да бъде специфицирано като обратно на друго.

- ***TransitiveProperty***: За всяко транзитивно свойство P е вярно, че ако двойките (x,y) и (y,z) са екземпляри на свойството P, то и двойката (x,z) също е екземпляр на P.

- ***SymmetricProperty***: Ако едно свойство P е симетрично, то винаги когато двойката (x,y) е екземпляр на P, двойката (y,x) също е екземпляр на P.

- ***FunctionalProperty***: Ако едно свойство е функционално, то това свойство има не повече от една стойност за всеки индивидуален обект.

Ограничения върху свойствата в OWL Lite

OWL Lite позволява да бъдат дефинирани ограничения върху това, как определени свойства могат да бъдат прилагани върху екземплярите на даден клас. Тези ограничения могат да бъдат специфицирани в контекста на конструкцията owl:Restriction.

- ***allValuesFrom***: Ограничението *allValuesFrom* се налага върху дадено свойство по отношение на някакъв клас. Означава, че по отношение на посочения клас свойството има асоциирано локално range ограничение.

Така, ако екземпляр на класа е свързан чрез това свойство с друг индивид, то за този индивид може да се изведе принадлежност към класа, посочен в локалното range ограничение.

Например за свойството hasDaughter може да е зададено ограничение *allValuesFrom* по отношение на класа Person със стойност Woman (ограничение, според което всички индивиди от класа Person могат да са свързани чрез свойството hasDaughter само с индивиди от клас Woman).

- ***someValuesFrom***: Ограничението *someValuesFrom* се налага върху дадено свойство по отношение на някакъв клас. Означава, че по отношение на този клас свойството има поне една стойност от определен тип (която е екземпляр на определен клас).

Допълнителни възможности на OWL DL и OWL Full

OWL DL и OWL Full използват един и същ речник, но в рамките на OWL DL се налагат множество ограничения върху използването на някои конструкции.

Например, OWL DL изисква сепарация на типовете (един клас не може да бъде същевременно индивидуален обект или свойство и едно свойство не може да бъде същевременно индивидуален обект или клас).

Освен това, OWL DL изисква всяко свойство да бъде или ObjectProperty, или DatatypeProperty: свойствата от първия тип са релации между екземпляри на класове, а свойствата от втория тип са релации между екземпляри на класове и литерали (данни) от типове на XML Schema.

- ***disjointWith:*** Класовете могат да бъдат специфицирани като взаимно чужди (disjoint). Например, Man и Woman могат да бъдат специфицирани по този начин. Взаимно чуждите класове не могат да имат общи индивидуални обекти (общи екземпляри).

- ***hasValue***: Определя изискване даденото свойство да има за стойност конкретен индивид. Например екземплярите на класа dutchCitizens може да бъдат характеризирани като тези хора, за които свойството Nationalty има стойност theNetherlands.

- ***oneOf***: Позволява дефиниране на класове чрез директно изброяване на техните екземпляри. Например класът daysOfTheWeek може да бъде описан чрез директно изброяване на принадлежащите му екземпляри Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday.

- ***unionOf, complementOf, intersectionOf***: OWL DL и OWL Full позволяват задаване на произволни булеви комбинации от класове и ограничения (чрез описание на обединение, разлика/допълнение, сечение).

- ***minCardinality***, ***maxCardinality***, ***cardinality***: OWL DL и OWL Full позволяват задаване на произволни неотрицателни цели числа като стойности на тези ограничения.

- ***Съставни класове***: За разлика от OWL Lite, който в много конструкции разрешава използване само на именувани класове (т.е. посочване само на имена на класове), OWL DL и OWL Full разрешават използване на съставни описания на класове. В допълнение OWL Full допуска класове да бъдат използвани като обекти/екземпляри на класове.

Системи, основани на знания - зимен семестър,
2022/2023 учебна година

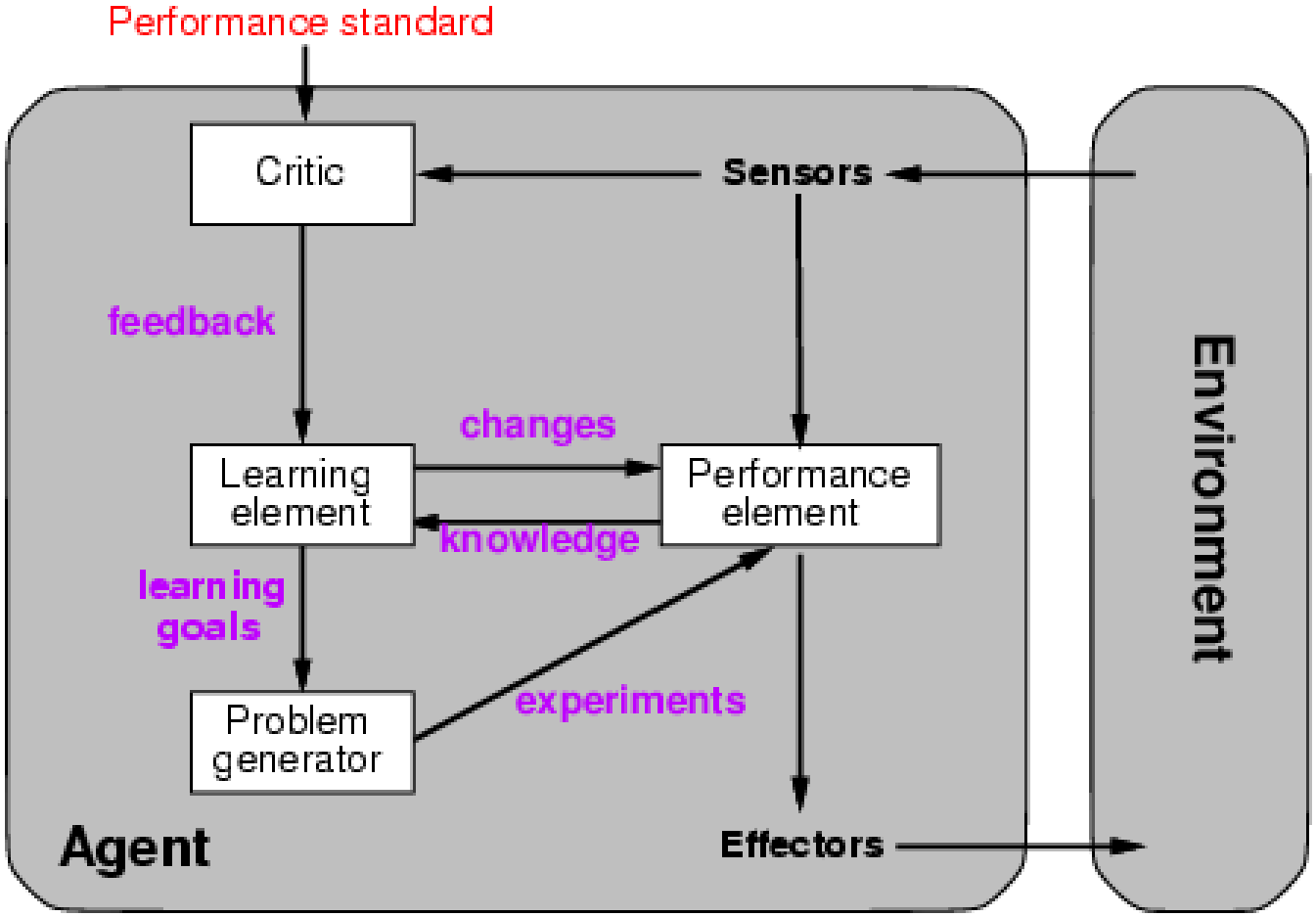
Тема 14:
Машинно самообучение

Същност на машинното самообучение

Х. Саймън: „Самообучението означава настъпване на такива промени в системата, които са адаптивни в смисъл, че позволяват на тази система при всеки следващ опит да извършва дадена работа по-ефективно и по-ефектно, отколкото при предишните опити“.

Р. Михалски: „Самообучението е конструиране или модифициране на различни представяния на предмета на дейност“.

Самообучаващ се агент:



Learning agent = performance element + learning element

Типове машинно самообучение в зависимост от характера на обратната връзка:

- Машинно самообучение (МС) с учител (наблюдавано МС, supervised machine learning)
- Машинно самообучение без учител (ненаблюдавано МС, unsupervised machine learning)

Самообучение чрез наизустяване

Представлява най-примитивен тип машинно самообучение (МС), аналог на ученето наизуст (зазубрянето) при човешкото обучение.

Идея. Запазват се резултати от работата на системата (условията на тежки, трудоемки задачи и получените решения) с цел използването им наготово в случай, че постъпи заявка за решаване на вече позната задача. В чистия си вид методът не предполага никаква допълнителна обработка на тези резултати. Оказва се, че при редица задачи такова запазване води до чувствително подобряване на работата на системата (повишаване на бързодействието, икономия на памет и др.).

Самообучение чрез макрооператори

Идея. Същата, както при самообучението чрез наизустяване – вариант на този тип МС, при който се предполага допълнителна обработка (обобщаване) на съхраняваните резултати от работата на системата. Използва се най-често в системи, свързани с планиране на действията.

Пример. Системата за планиране STRIPS има обучаваща подсистема за натрупване на резултати от вече извършена от нея работа под формата на макрооператори. Всеки макрооператор представлява обобщен план за решаване на даден тип задачи и се състои от *предусловия* (обобщават съществена част от началното състояние при решаването на конкретна задача за планиране), *тяло* (обобщение на конструирания от системата план) и *следусловия* (обобщение на целта при решаването на конкретна задача за планиране).

Самообучение чрез запомняне

Идея. Подход за самообучение с учител, при който целта е на базата на множество зададени от учителя решени примери (т. нар. *обучаващи примери*) за обекти, принадлежащи на *множество от известни класове*, да се класифицира определен *тестов пример*.

Най-използваният метод за самообучение чрез запомняне е *методът на най-близкия съсед* (NN – Nearest Neighbour). При този метод класификацията на тестовия пример зависи от степента на неговото сходство с единствен пример на понятие (клас) – този, който се намира на най-малко разстояние от него (неговия най-близък съсед).

Приема се, че тестовият пример принадлежи на този клас (това понятие), чийто представител е най-близо до него.

Друг често използван в практиката метод за самообучение чрез запомняне е *методът на k най-близки съседи* (k-NN). Числото k определя броя на най-близките екземпляри на понятия (класове) измежду обучаващите примери, които участват в определянето на решението за класификация на тестовия пример. Тестовият пример се класифицира в съответствие с (като представител на) класа, който се среща най-често сред неговите k най-близки съседи. Ако повече от един клас се среща най-често сред най-близките k съседи на тестовия пример, той обикновено се класифицира в съответствие с класа на най-близкия свой съсед сред конкуриращите се класове (най-близките негови k съседи).

Самообучение чрез уточняване на параметрите

Идея. Много системи с изкуствен интелект използват оценяващи (или разделящи) функции, които представляват комбинации от стойности на подходящо подбрано множество от параметри (признаци, фактори), като всеки параметър участва в оценяващата функция със съответен коефициент (тегло). При проектирането и първоначалното програмиране на такива системи обикновено е трудно априори да се определят правилно теглата на всички фактори, които участват във формулировката на оценяващата функция. Един възможен подход за правилното определяне на теглата на отделните фактори е свързан с включването на средства за модифициране (доуточняване) на тези тегла на основата на натрупания опит от работата на системата с текущия вариант на оценяваща функция.

Пример: програма на Samuel за игра на шашки (МС чрез наизустяване + МС чрез уточняване на параметрите)

Това е един от най-често използваните методи за МС в областта на разпознаването на образи и при невронните мрежи.

Самообучение чрез съвети

Идея. Знанията се придобиват от учител (евентуално учебник или др.) и съответната обучаваща подсистема трансформира тези знания във вид, който е използваем от обработващата подсистема. С други думи, учителят избира съдържанието, структурата и представянето на знанията (съвета), които иска да добави към съществуващите знания на системата, а обучаващата подсистема има за задача синтактичното преобразуване (преформулировката) на знанията, получени от учителя, във вид, който е достъпен за съответния обработващ (използващ знанията) модул.

Пример 1. В шахмата съвет от рода на „Стремете се да контролирате централната част на дъската“ е практически неизползваем за съответната програма за игра на шах, ако не съществува обучаваща програма, която да използва този съвет, като на негова основа коригира съответната оценяваща позициите функция (например като добави нов фактор за отчитане на броя на централните полета, които се контролират, т.е. могат да се атакуват от фигурите на съответния играч).

Пример 2. Програмата TEIRESIAS, която играе ролята ня интерфейс за получаване на съвети към популярната експертна система MYCIN. TEIRESIAS извършва специализации и обобщения на базата на съвети, давани от учител, като основава работата си на две прости идеи: разчита на воден от меню диалог, който прави нейната вътрешна структура максимално ясна за учителя; когато е необходим вход на естествен език, програмата се консултира с учителя, за да се убеди, че е превела (разбрала) правилно входния текст.

Самообучение чрез примери

Идея. Това е тип МС (частен случай на т. нар. *индуктивно самообучение*), което обикновено се прилага при системи, предназначени за решаване на задачи, свързани с класификация на обекти.

Класификацията е процес, при който по даден обект (описанието на даден обект) се получава името на класа, към който принадлежи този обект. Класификацията е важен етап от решаването на много задачи от областта на ИИ. При създаването на системи за решаване на такива задачи преди всичко трябва да бъдат дефинирани класовете, с които ще се работи. Често е много трудно да се даде пълна и вярна дефиниция на отделните класове в разглежданата област. Поради това е полезно създаването на самообучаващи се програмни системи, които могат сами да изграждат дефинициите на класовете от предметната област. Задачата за изграждането на дефинициите на класове е известна като *изучаване на понятия* (concept learning). МС чрез примери е най-популярният метод за изучаване на понятия.

Характерно за този тип МС е използването на множество от т. нар. ***обучаващи примери***, които могат да бъдат от два типа: *положителни* и *отрицателни* обучаващи примери.

Ще разгледаме накратко два популярни подхода при МС чрез примери: подхода на т. нар. *отделяне* и подхода на т. нар. *покриване.*

Подход на т. нар. ***отделяне***: целта е *примерите за даден клас да се отделят* от тези за другите класове. Типичен представител на такава *разделяща стратегия* е подходът на *класификационните дървета*.

Построяване на класификационни дървета

Класификационно дърво (КД): Всеки *възел* в едно КД представя даден атрибут, срещащ се в примерите. *Дъгите*, излизащи от даден възел, представят различните възможни стойности на този атрибут. Всеки *лист* на дървото определя класификацията на даден пример, като стойностите на съответните атрибути са зададени по пътя от корена към този лист.

В случай на едно понятие листата се маркират с “+” (или T, True) за положителните примери и с “-“ (или F, False) – за отрицателните примери.

Примерна задача. Да се вземе решение дали да се чака за маса в ресторант на базата на конкретните стойности на следните атрибути, които показват:

- ✓ *Alternate*: дали наблизо има друг подходящ ресторант
- ✓ *Bar*: дали ресторантът разполага с удобно място за изчакване
- ✓ *Fri/Sat*: дали денят е петък или събота
- ✓ *Hungry*: дали сме гладни
- ✓ *Patrons*: колко хора има в ресторанта (възможни стойности: None, Some, Full)

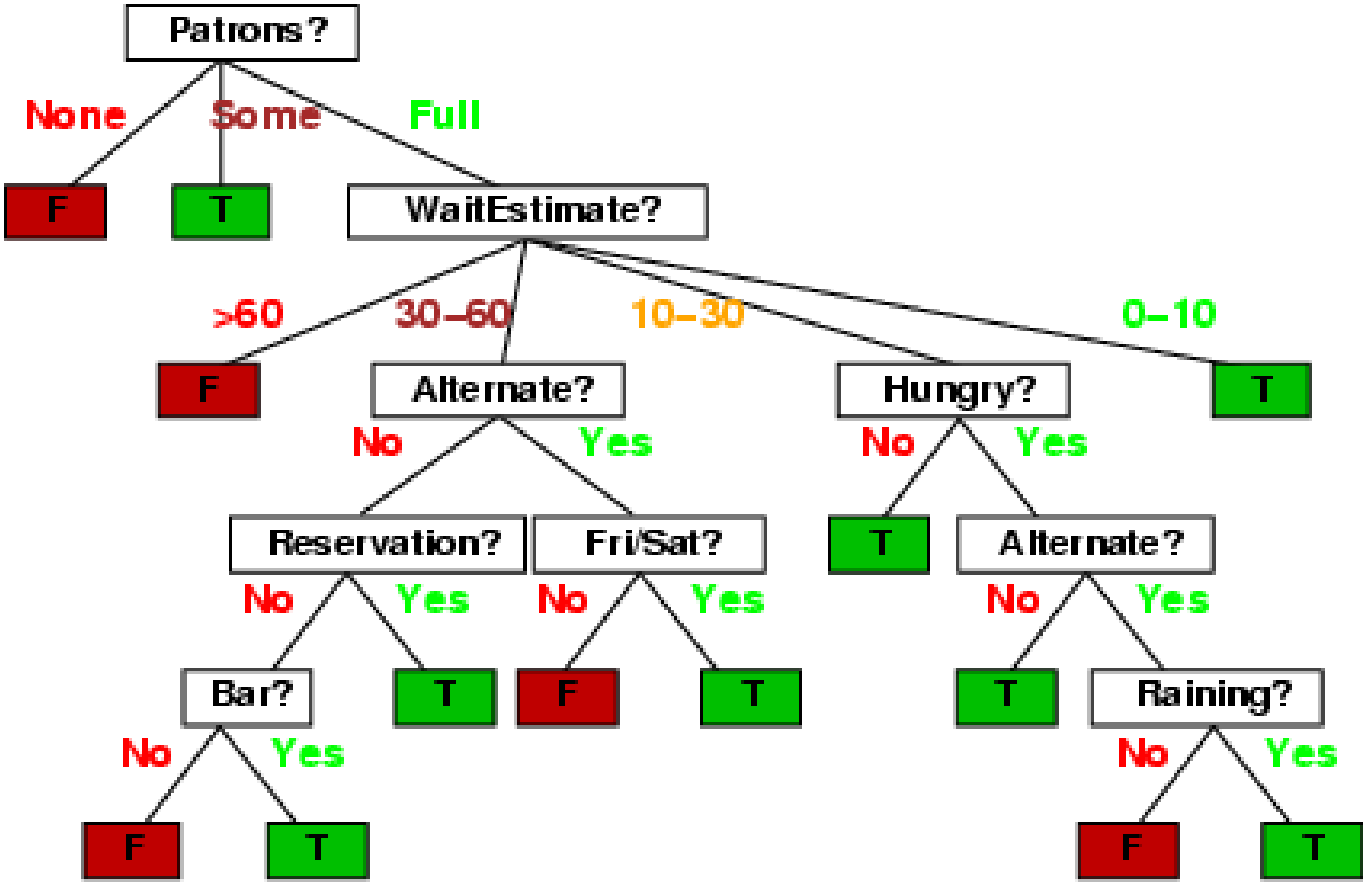
- ✓ *Price*: какво е нивото на цените в ресторанта ($, $$, $$$)
- ✓ *Raining*: дали навън вали
- ✓ *Reservation*: дали имаме предварително направена резервация
- ✓ *Type*: типът на ресторанта (френски, италиански и т.н.)
- ✓ *WaitEstimate*: предполагаемото време за чакане според персонала (0-10 минути, 10-30, 30-60, >60)

- Примерите се описват чрез подходящи стойности на атрибутите (булеви, дискретни, непрекъснати)
- Примерни ситуации, при които потребителят ще реши да чака (или да не чака) за маса:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- Примерите се разделят на положителни (+/T) и отрицателни (-/F)

Примерно класификационно дърво:



```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

Забележка. Ако няма останали атрибути за избор, но класификацията на примерите все още не е приключила (има останали както положителни, така и отрицателни обучаващи примери), това означава, че съществува проблем. Останали са обучаващи примери, които имат еднакви стойности на атрибутите, но се класифицират по различни начини. Такава ситуация може да възникне, когато данните са некоректни (зашумени), когато избраните атрибути са недостатъчни или когато областта е недетерминирана.

В тези случаи функцията DTL връща резултат, който се определя от функцията MODE.

Естествено, видът на полученото класификационно дърво зависи от реда на ***избор на атрибути***.

При конструирането на класификационни дървета целта е да се построи *„компактно“ дърво с минимален брой разделящи възли* (т.е. целта е да се минимизира необходимият брой тестове). Така за предпочитане са плитките и силно разклонени дървета.

Постигането на тази цел се определя от избора на „добри“ атрибути. Критерий за качеството на даден атрибут е способността му да разделя множеството от примери на групи от еднородни примери. Формално това най-често се прави с използване на т. нар. *информационен критерий*.

Избор на „най-добрия“ атрибут

Основната идея на информационния критерий е да се изчисли *ентропията* на разпределението на обучаващите примери по класове, получено при разбиването на множеството от примери чрез стойностите на всеки атрибут. Най-добрият атрибут е този, при който се получава най-малка стойност на ентропията.

Оценката на ентропията (информационното съдържание) се основава на теорията на информацията на Шенън. Ентропията е обща оценка на нивото на „изненада“ или „неопределеност“ при получаването на n различни съобщения и се изчислява по формулата

$I = - \sum P_i \log_2 P_i$, където P_i ($i = 1, \ldots, n$) са вероятностите на отделните съобщения.

Предполага се, че всички примери от обучаващата извадка се срещат с една и съща вероятност. Тогава на базата на горната формула може да се оцени информационното съдържание (ентропията) $I(E)$ на даденото множество от обучаващи примери E. Същата оценка се отнася и за всяко класификационно дърво, което съответства на (покрива) това множество.

Ентропията може да се използва за оценка на качеството на избора на конкретен атрибут за разделящ възел в класификационното дърво. За тази цел се използва разликата между общото количество информация в дървото преди избора на атрибута и остатъка от информация, необходима за довършване на дървото с корен избрания атрибут.

Да предположим, че атрибутът *A* с *n* възможни стойности е избран за разделящ възел. Тогава той става корен на *n* поддървета, разделящи текущото множество от обучаващи примери *E* на *n* подмножества: E_1, E_2, ... , E_n. Може да се покаже, че *информацията, необходима за довършване на дървото след избора на A като корен*, е

$$R(A) = \sum_{i=1}^{n} \frac{|E_i|}{|E|} I(E_i)$$

Тогава информационната печалба от избора на атрибута A се пресмята по формулата

$$gain(A) = I(E) - R(A)$$

На всяка стъпка от работата на алгоритъма се избира атрибутът, който носи най-голяма информационна печалба.

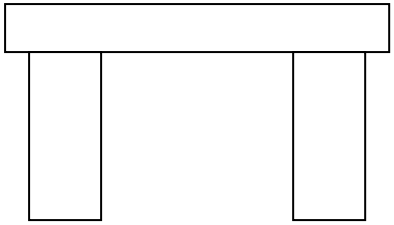
Подход на т. нар. ***покриване***: целта при построяване на описанията е те да *покриват примерите* (положителните примери) за дадения клас. По-точно, за всеки клас се задава (пълно) множество от положителни обучаващи примери (описания на обекти, принадлежащи на класа) и множество от отрицателни обучаващи примери (описания на обекти, които не принадлежат на класа, но приличат на обекти от положителните примери). Извършва се *обобщаване* на положителните примери (по такъв начин, че полученото описание на класа да включва като частни случаи описанията на всички положителни примери) и *ограничаване* на отрицателните примери (по такъв начин, че описанието на класа да изключва описанията на отрицателните примери).

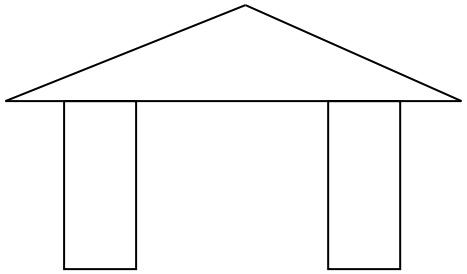
Пример: програмата ARCHES на P. Winston за класификация на обекти от света на кубчетата.

ARCHES използва структурно описание на дефинициите на понятията от света на кубчетата. Тя съдържа множество процедури за въвеждане на стилизирани рисунки на съответните обучаващи примери, които анализират фигурите от примерите и конструират подходящи семантични мрежи – структурни описания на различните обекти.

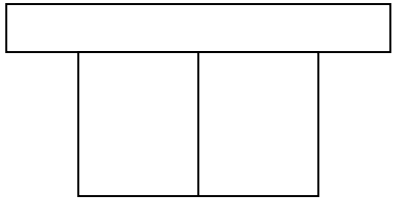
Обучаващи примери за понятието „арка“ (arch):

положителни примери



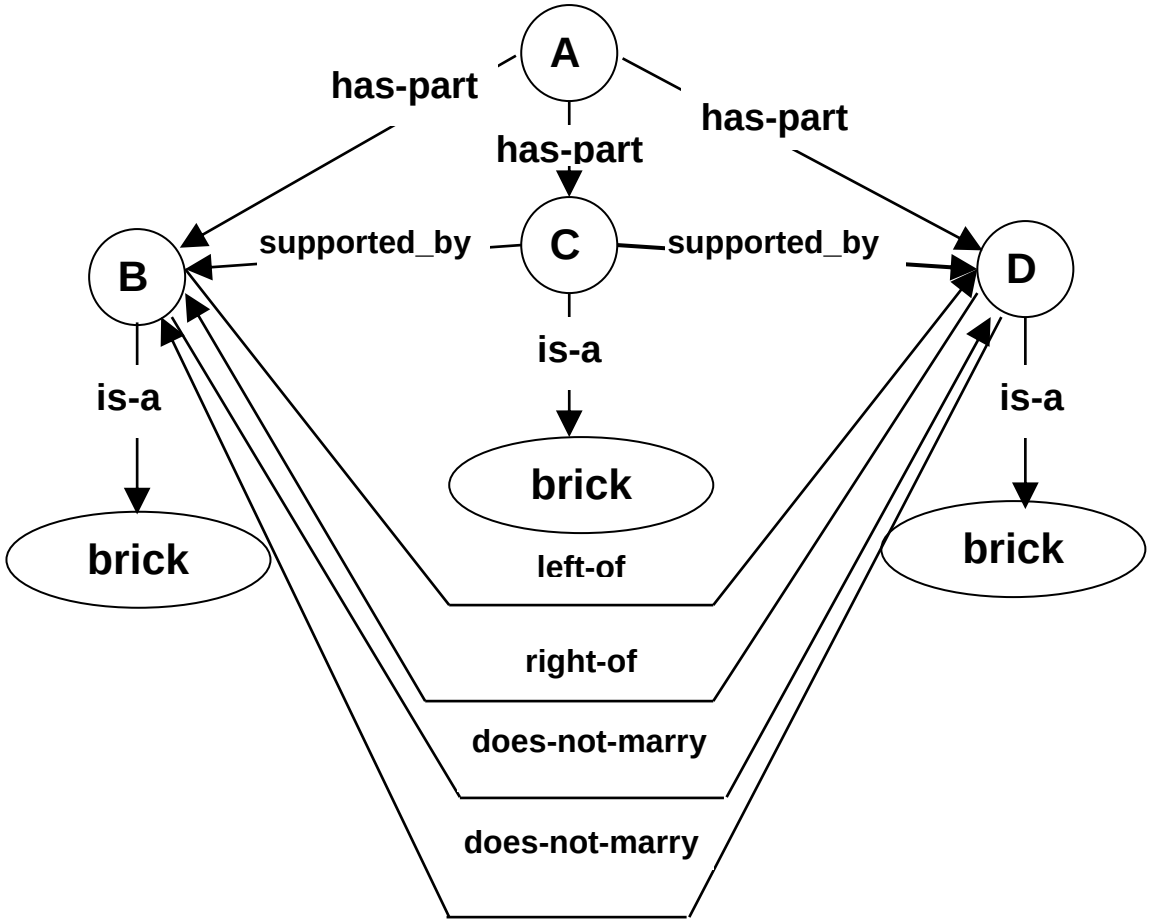


отрицателни примери

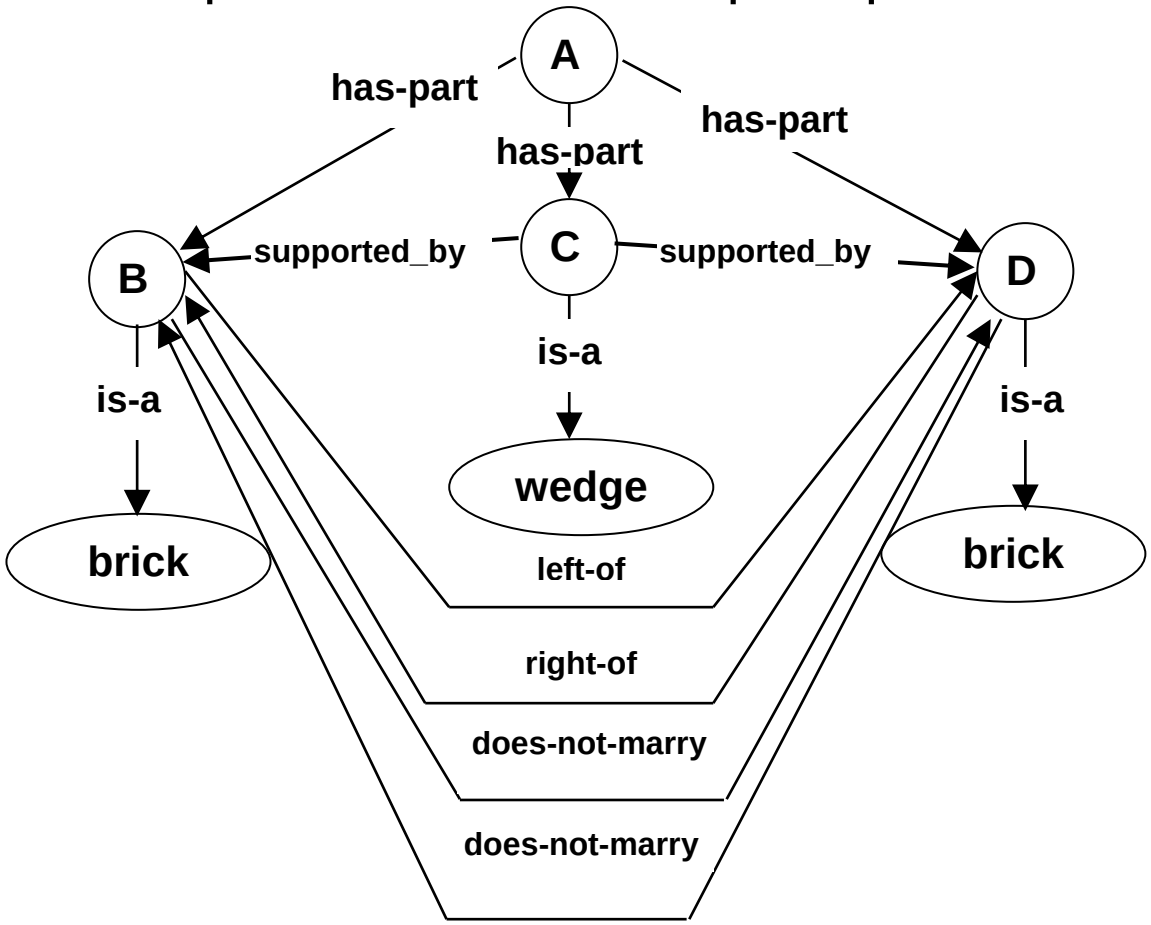


Създаване на описанието на изучаваното понятие (арка): чрез обобщение на описанията на положителните примери (което да покрива множеството на положителните обучаващи примери) и ограничаване на отрицателните обучаващи примери (така че обобщеното описание на положителните примери да изключва отрицателните). За целта се използват множество подходящи евристики.

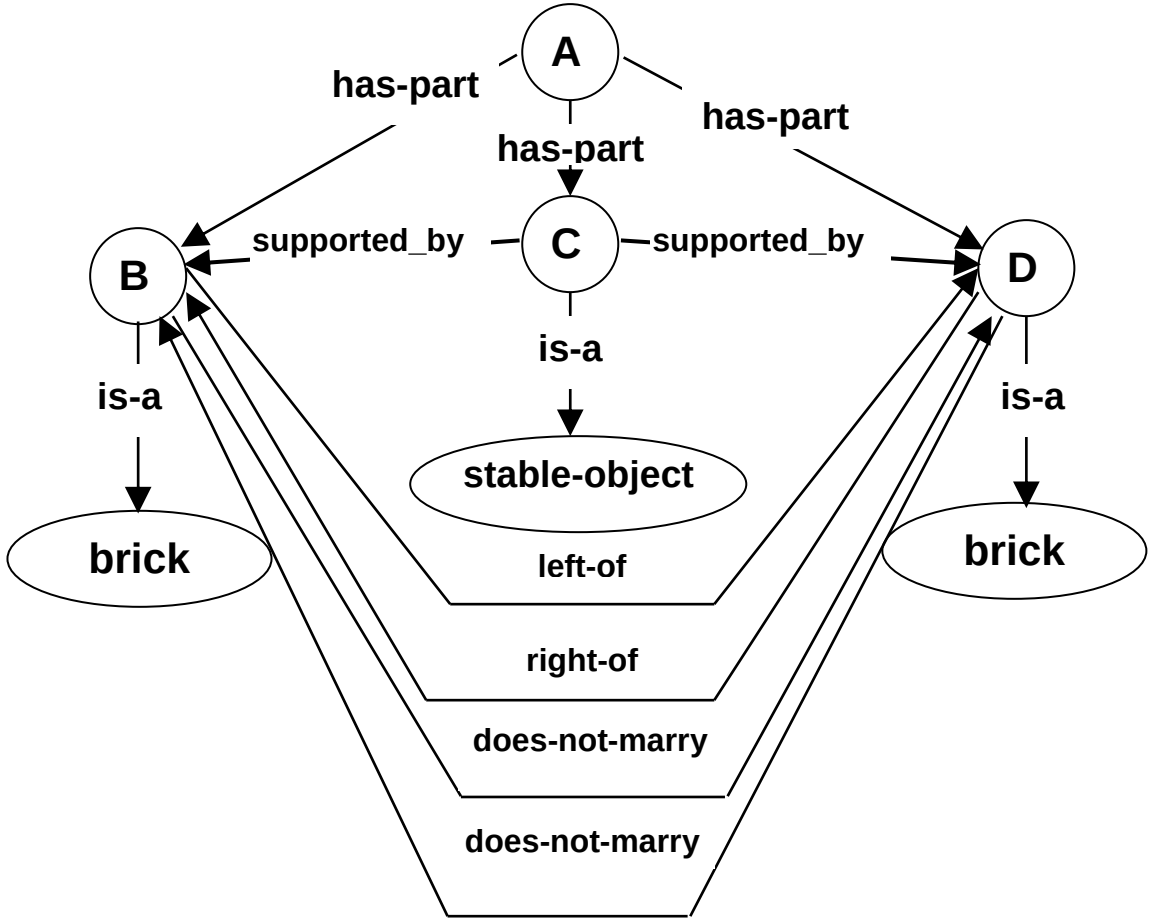
Описание на първия положителен пример:



Описание на втория положителен пример:

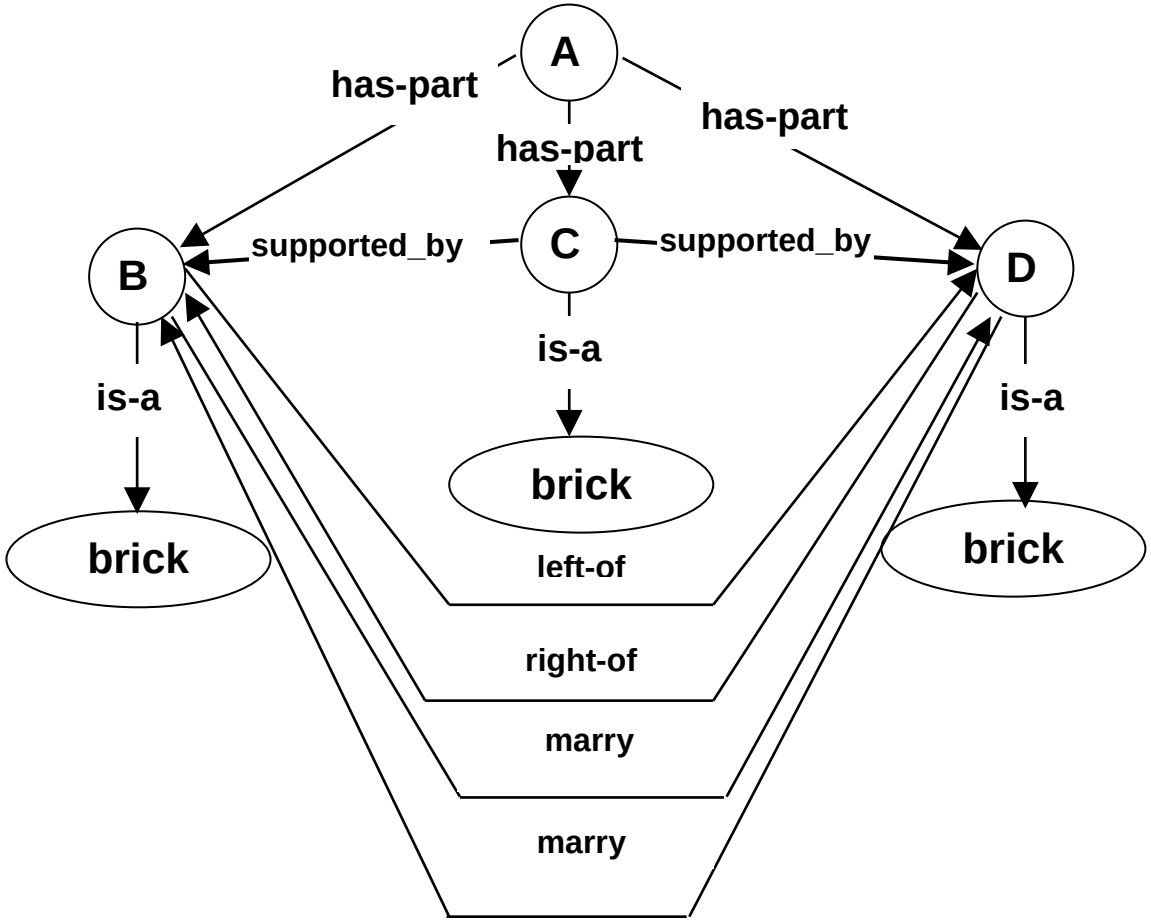


Обобщение на описанията на двата положителни примера:



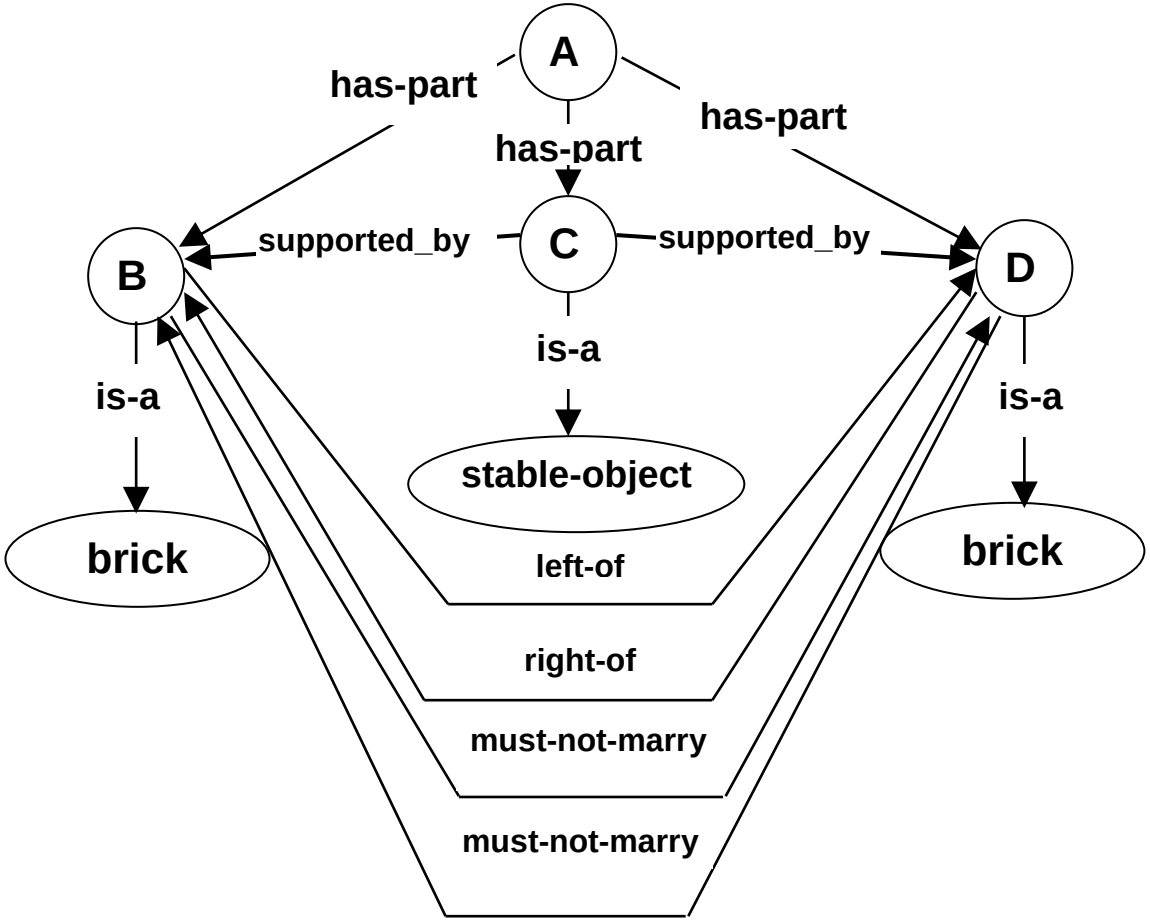
Забележка. Предполага се, че stable-object е най-близкият общ предшественик в йерархията на геометричните форми на формите brick и wedge. С други думи, обобщаването на положителните примери е свързано с минимално обобщаване на съответните геометрични форми.

Описание на отрицателния пример:



Огранчаване на отрицателния пример: чрез засилване на името на връзката “does-not-marry” до “must-not-marry” (за положителните примери е изпълнено “does-not-marry”, за отрицателния е изпълнено обратното, следователно валидната за положителните примери релация е решаваща за дефинирането на изучаваното понятие).

Окончателно описание на понятието „арка“:



Самообучение чрез обяснение

Идея. При МС чрез примери съществено е честото използване на голям брой обучаващи примери (положителни и отрицателни). Тук се използва един обучаващ пример и на основата на допълнителни знания за предметната област се конструира обяснение защо той е пример за разглежданото понятие. Чрез обобщение на обяснението се получава описанието на изучаваното понятие.

При самообучението чрез обяснение съответната обучаваща програма използва следните видове данни/знания:

- *Целево понятие* (описание на високо равнище на целевото понятие в термините на предметната област, но без да е изпълнен т. нар. критерий за операционалност);
- *Критерий за операционалност (конструктивност)* – описание или списък на понятията, в чиито термини трябва да бъде формирано описанието на целевото понятие;
- *Обучаващ пример* – описание на обект от предметната област, който е положителен пример за целевото понятие;
- *Теория за предметната област* – множество от правила, които описват необходимите знания за класификация на обектите в предметната област (в термините на критерия за операционалност).

На основата на тези данни и знания обучаващата програма конструира обяснение (доказателство например чрез използване на обратен извод върху теорията за предметната област) защо обучаващият пример е пример за изучаваното (целевото) понятие, което е формулирано в термините на критерия за операционалност. Чрез обобщение на това обяснение се получава търсеното описание на целевото понятие.

Пример (Митчел и др., 1986 г.): *обучение на понятието „чаша“*. Съгласно отбелязаното по-горе, съответната система разполага с:

1. *Обучаващ пример – обект23*

собственик(обект23, Джон) ^ Е(обект23, лек) ^
цвят(обект23, кафяв) ^ има-част(обект23, дръжка16) ^
има-част(обект23, дъно19) ^ има-част(обект23, вдлъбнатина12) ^
.....

Очевидно някои от характеристиките (признаците) на *обект23* са по-съществени за определянето му като чаша от други. Изолирането на действително важните признаци от гледна точка на описанието на съответното понятие при този метод става с помощта на знанията (теорията) за предметната област.

2. *Целево понятие – чаша*: предметът *x* е чаша тогава и само тогава, когато има свойствата *премествам*, *отворен* и *стабилен-съд*.

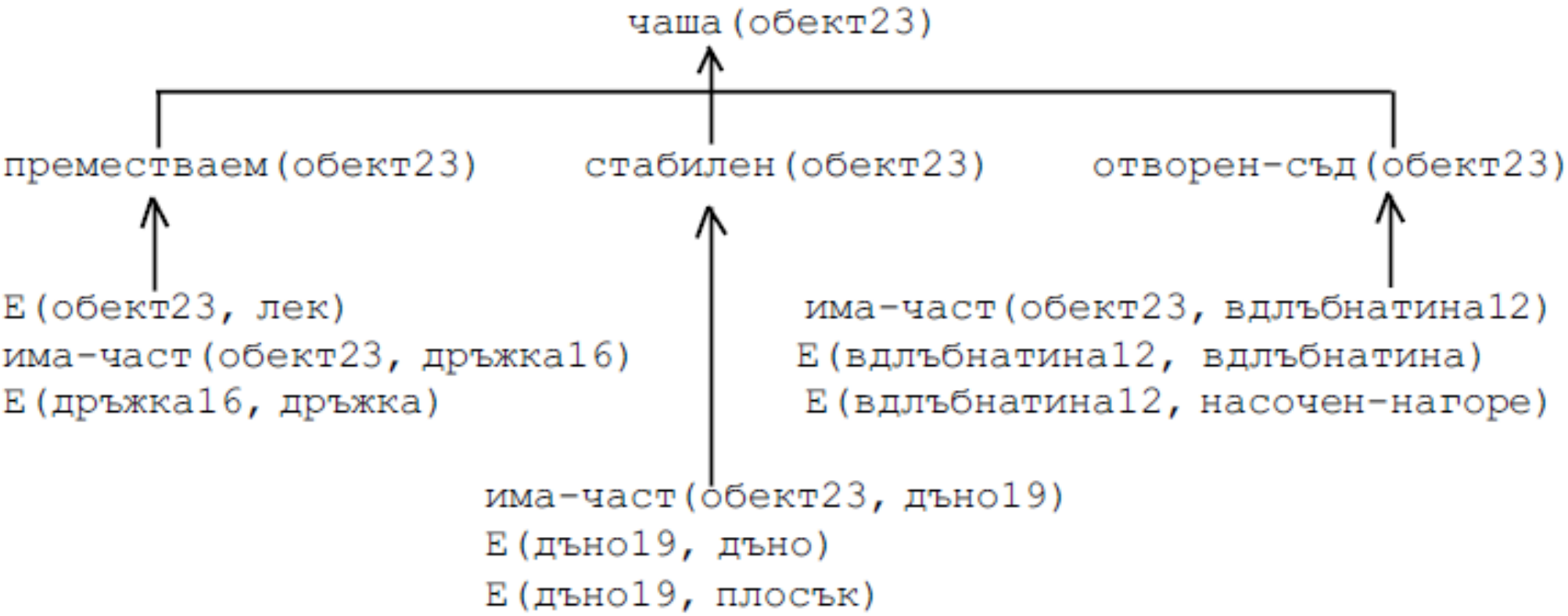
3. *Критерий за конструктивност (операционалност)*: описанието на понятието трябва да бъде изразено в прости структурни термини (*лек*, *плосък*, *има-част* и т.н.).

4. Теория (знания) за предметната област

…..
Е(х, лек) ^ има-част(х, у) ^ Е(у, дръжка) ⇒ премествӑем(х)
има-част(х, у) ^ Е(у, дъно) ^ Е(у, плосък) ⇒ стабилен(х)
има-част(х, у) ^ Е(у, вдлъбнатина) ^ Е(у, насочен-нагоре) ⇒
отворен-съд(х)
…..

Задачата, която решава обучаващата система, е формиране на конструктивно описание на понятието *чаша*.

Първата стъпка е построяване на обяснение на това, защо *обект23* е чаша. Това може да стане чрез построяване на доказателство, както е показано на следващата фигура. За целта могат да се използват стандартните методи за извод при продукционните системи.


чаша(обект23)
премества(обект23)
стабилен(обект23)
отворен-съд(обект23)
Е(обект23, лек)
има-част(обект23, дръжка16)
Е(дръжка16, дръжка)
има-част(обект23, дъно19)
Е(дъно19, дъно)
Е(дъно19, плосък)
има-част(обект23, вдлъбнатина12)
Е(вдлъбнатина12, вдлъбнатина)
Е(вдлъбнатина12, насочен-нагоре)

Горното доказателство изолира съществените признаци на обекта, представен от обучаващия пример. То е и основа за извършване на ***обобщение***. Ако обединим всички използвани предположения и заменим константите с променливи, получаваме следното описание на понятието *чаша*:

има-част(x, y) ^ E(y, вдлъбнатина) ^ E(y, насочен-нагоре) ^
има-част(x, z) ^ E(z, дъно) ^ E(z, плосък) ^
има-част(x, w) ^ E(w, дръжка) ^ E(x, лек)

Това описание удовлетворява критерия за конструктивност и може да бъде използвано например от робот за класификация на обекти от предметната област.

Най-съществено за обучението чрез обяснение е използването на знания за предметната област. Тези знания по принцип са достатъчни за извършването на класификация на обектите от областта. Използването на обучаващ пример за разглежданото понятие дава възможност за по-голяма ефективност и конструктивност. Примерът насочва и ограничава частта от знанията за областта, които са приложими в конкретния случай. В противен случай в процеса на извод ще се използва цялата БЗ за предметната област.

Системи, основани на знания - зимен семестър,
2022/2023 учебна година

Тема 15:
Бейсов класификатор. Класификация на текстове

Често в областта на машинното самообучение се решават задачи, свързани с определяне на най-добрата хипотеза от някакво пространство на хипотези H при зададени обучаващи данни D. Един от начините да се определи значението на често използвания термин „най-добра хипотеза“ е да се счита, че най-добра е *най-вероятната* хипотеза при зададени данни D и налични в момента знания за априорните вероятности на различните хипотези в H.

Теоремата на Бейс дава директен метод за изчисляване на вероятностите на отделните хипотези. По-точно, тази теорема позволява да изчислим вероятността на една хипотеза въз основа на нейната априорна вероятност, вероятността да се наблюдават съответните данни при наличието на хипотезата и вероятността на самите данни.

Означения

С *P(h)* означаваме началната вероятност на предположението, че хипотезата *h* е вярна, преди да наблюдаваме каквито и да е обучаващи данни. *P(h)* се нарича *априорна вероятност на h* и може да отразява всякакви основни знания, с които разполагаме, за шансовете на *h* да бъде коректна хипотеза. Ако нямаме никакви априорни знания за възможни хипотези, можем просто да определим една и съща априорна вероятност за всяка от тях.

С *P(D)* означаваме априорната вероятност за наблюдаване на обучаващите данни *D* (т.е. вероятността на *D* без да са налице никакви знания за това, коя от хипотезите е вярна).

P(D|h) означава вероятността да наблюдаваме данните *D* при условие, че хипотезата *h* е вярна. В общия случай *условната вероятност P(x|y)* означава вероятността да се случи някакво събитие *x* при условие, че вече се е случило събитието *y*.

В машинното самообучение интерес представлява вероятността *P(h|D)*, че хипотезата *h* е вярна при наблюдавани обучаващи данни *D*. Тази вероятност се нарича *апостериорна вероятност на h*, тъй като отразява степента на нашата убеденост, че *h* е вярна *след* наблюдаване на *D.* Апостериорната вероятност отразява влиянието на данните *D*, за разлика от априорната вероятност *P(h)*, която не зависи от никакви данни.

Теоремата на Бейс е основата на Бейсовите методи за самообучение, тъй като осигурява начин за изчисляване на апостериорната вероятност *P(h|D)* от известни априорни вероятности *P(h)* и *P(D)*, както и от *P(D|h)*:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Както се вижда, *P(h|D)* нараства с нарастване на *P(h)* и *P(D|h)* и намалява с увеличаването на *P(D)*, тъй като колкото по-вероятно е, че данните *D* се наблюдават независимо от *h*, толкова по-малка поддръжка дава *D* за съществуването (валидността) на *h*.

В много задачи от областта на МС разглеждаме определено множество от възможни хипотези *H* и се интересуваме от намирането на най-вероятната от тях при наблюдавани данни *D* (или от максимално вероятна – ако има няколко). Такава максимално вероятна хипотеза се нарича *максимална апостериорна* (MAP) хипотеза. Можем да определим MAP хипотезата с използване на теоремата на Бейс:

$$h_{MAP} = argmax_{h \in H} P(h|D) = argmax_{h \in H} P(D|h)P(h)$$

В горното равенство вероятността *P(D)* е игнорирана, тъй като тя е постоянна и не зависи от *h.*

В някои случаи може да се приеме, че всички хипотези в *H* имат еднаква априорна вероятност. В такива случаи в горната формула има смисъл да разглеждаме само множителя *P(D|h)*, който често се нарича *възможност* (likelihood) на данните *D* при зададена *h*. Всяка хипотеза, максимизираща *P(D|h)*, се нарича *максимално възможна* (ML) хипотеза:

$$h_{ML} = argmax_{h \in H} P(D|h)$$

Често в машинното самообучение *H* се свързва с пространство от възможни целеви функции, подлежащи на научаване, а *D* – с обучаващи примери за някаква целева функция. Теоремата на Бейс е приложима и в по-общ контекст, където *H* се разглежда като множество от взаимно изключващи се предположения, чиято сумарна вероятност е равна на 1.

Досега разисквахме въпроса „каква е най-вероятната хипотеза при зададени обучаващи данни?“. Много често в практиката е по-важен въпросът „каква е най-вероятна *класификация на нов пример* при зададени обучаващи данни?“ Макар да изглежда, че отговорът на този въпрос може да бъде получен просто чрез прилагане на MAP хипотеза към новия пример, на практика е възможен по-добър подход.

Оптимален Бейсов класификатор

Нека например да разгледаме пространство от хипотези, съдържащо три хипотези *h1, h2* и *h3*. Нека техните апостериорни вероятности при зададени обучаващи данни са съответно 0.4, 0.3 и 0.3. Следователно *h1* е MAP хипотеза. Да предположим, че постъпва нов пример, който се класифицира като положителен от *h1* и като отрицателен от *h2* и *h3*. Отчитайки всички хипотези, вероятността, че примерът е положителен, е 0.4, а че примерът е отрицателен – 0.6. Най- вероятната класификация в този случай е различна от класификацията, генерирана от MAP хипотезата.

В общия случай най-вероятната класификация на даден нов пример се получава от комбиниране на предсказанията на всички хипотези, претеглени от техните апостериорни вероятности. Ако за възможна класификация на новия пример може да приеме една от стойностите v_j от множеството V, то вероятността $P(v_j|D)$ за коректна класификация на новия пример като v_j се получава с използване на формулата

$$P(v_i|D) = \sum_{h_j \in H} P\left(v_j|h_j\right) P(h_j|D)$$

Оптималната Бейсова класификация на новия пример е стойността v_j, за която $P(v_j|D)$ е максимална:

$$v_j \equiv argmax_{v_j \in V} \sum_{h_j \in H} P(v_j|h_j) P(h_j|D)$$

Всяка система, която класифицира в съгласие с горната формула, се нарича *оптимален Бейсов класификатор*. Никой друг метод за класификация, използващ същото пространство на хипотези и същите априорни знания, *не може да подобри* (като средно) този метод. Той максимизира вероятността, че новият пример е класифициран правилно при зададените налични данни, пространството от хипотези и априорните вероятности на хипотезите.

Макар че оптималният Бейсов класификатор постига най-доброто поведение, което може да бъде постигнато при наличните обучаващи данни, неговото прилагане е много скъпо от изчислителната гледна точка. Това се дължи на необходимостта да бъдат изчислени апостериорните вероятности на всички възможни хипотези в H, след което трябва да бъдат комбинирани предсказанията на всички хипотези, за да бъде класифициран разглежданият нов пример.

Наивен Бейсов класификатор

Да разгледаме случая, когато примерът x за научаваното понятие се описва с конюнкция от атрибутни стойности, а целевата функция $f(x)$ приема дискретни стойности от дадено крайно множество V.

Дадени са множество от обучаващи примери D и нов пример, подлежащ на класификация: $x = <a_1, \ldots, a_n>$. Алгоритмът за обучение трябва да предскаже стойността на целевата функция за този пример, т.е. да класифицира този пример.

Съгласно Бейсовия подход класифицирането на примера се свежда до избор на най-вероятната стойност на целевата функция v_{MAP} при зададените атрибутни стойности $<a_1, \ldots, a_n>$, описващи примера:

$$v_{MAP} = argmax_{v_j \in V} P(v_j | a_1, \ldots, a_n)$$

Използвайки теоремата на Бейс, получаваме:

$$v_{MAP} = argmax_{v_j \in V} \frac{P(a_1, \ldots, a_n | v_j) P(v_j)}{P(a_1, \ldots, a_n)} =$$

$$argmax_{v_j \in V} P(a_1, \ldots, a_n | v_j) P(v_j) \quad (1)$$

Сега трябва да опитаме да оценим двата терма в равенство (1) на базата на обучаващите данни. $P(v_j)$ е лесно да се оцени чрез изчисляване на честотата на срещане на всяка стойност v_j на целевата функция в множеството от обучаващи данни. Но количествена оценка на другия терм по този начин е практически невъзможна, освен ако не разполагаме с много голямо множество от обучаващи данни.

Проблемът е в това, че броят на термовете от този вид е равен на броя на всички възможни примери (комбинации от атрибутните стойности), умножен по броя на възможните стойности на целевата функция. Следователно, за да получим едно надеждно приближение за подобна оценка, трябва да прегледаме многократно всеки възможен пример в пространството от примери.

Един възможен начин за избягване на тези усложнения е да предположим, че атрибутните стойности са *условно независими при зададената стойност на целевия атрибут*, т.е. че за дадената стойност на целевата функция вероятността за съществуване на конюнкцията на атрибутните стойностите $a_1, \ldots, a_n$ е равна на произведението на вероятностите на отделните атрибутни стойности:

$$P(a_1, \ldots, a_n | v_j) = \prod_{i=1}^{n} P(a_i | v_j)$$

Заместваме този терм в (1) и стигаме до подхода, известен като *наивен Бейсов класификатор:*

$$v_{NB} = argmax_{v_j \in V} P(v_j) \prod_{i=1}^{n} P(a_i | v_j) \quad (2)$$

Ще обърнем внимание, че в случая на наивния Бейсов класификатор броят на различните термове $P(a_i|v_j)$, чиито стойности трябва да бъдат оценени чрез обучаващите данни, е равен на броя на възможните атрибутни стойности, умножен по броя на възможните стойности на целевия атрибут – това число е значително по-малко от броя на термовете, които са необходими за оценяването на $P(a_1, \ldots, a_n \mid v_j)$.

Пример: Научаване на класификация на текстове

В качеството на илюстративен пример ще разгледаме един общ алгоритъм за научаване на класификация на текстове, базиран на наивния Бейсов класификатор.

Нека разгледаме пространството *X*, съдържащо всички възможни *текстови документи* (т.е. всички възможни низове от думи и пунктуационни знаци с произволна дължина). Дадени са обучаващи примери за някаква неизвестна целева функция *f(x),* която приема стойности от дадено крайно множество *V*. Задачата е да се научим от тези обучаващи примери да предсказваме стойността на целевия атрибут за нови текстови документи. За илюстрация ще разгледаме класификацията на документите само на две групи – *интересни* и *неинтересни*.

Най-напред е необходимо да бъдат решени два основни въпроса:

- как ще представяме произволен текстов документ в термините на атрибутните стойности;
- как да бъдат оценени вероятностите, необходими за работата на наивния Бейсов класификатор.

Предложеният подход е много прост: при зададен текстов документ, съдържащ например текст на английски език, се дефинира по един атрибут за всяка отделна позиция на дума в текста (за всеки конкретен пореден номер на дума в текста), а като стойност на този атрибут – съответната английска дума, намираща се на конкретната позиция.

Например цитираният по-долу параграф може да се опише със 111 позиции на думите. Стойността на първия атрибут е думата ”our”, на втория – “approach”, и т.н.

Our approach to representing arbitrary text documents is disturbingly simple: Given a text document, such as this paragraph, we define an attribute for each word position in the document and define the value of that attribute to be the English word found in that position. Thus, the current paragraph would be described by 111 attribute values, corresponding to the 111 word positions. The value of the first attribute is the word "our," the value of the second attribute is the word "approach," and so on. Notice that long text documents will require a larger number of attributes than short documents. As we shall see, this will not cause us any trouble.

Следователно, броят на необходимите атрибути е равен на броя на думите в текста, т.е. по-дългите текстови документи изискват по-голям брой атрибути от по-късите документи.

След като сме избрали подходящо кодиране на текстовите документи, вече можем да приложим наивния Бейсов класификатор. Да предположим, че разполагаме с набор от 700 документа, които са класифицирани като „*неинтересни*“, и други 300 документа, които са класифицирани като „*интересни*“. Предоставен ни е нов документ, който трябва да класифицираме като „интересен“ или „неинтересен“.

Нека предположим, че този документ е съставен от цитирания по-горе параграф.

В такъв случай от равенство (2) получаваме

$$v_{NB} = argmax_{v_j \in \{\text{интересни,неинтересни}\}} P(v_j) \prod_{i=1}^{111} P(a_i | v_j) =$$

$$argmax_{v_j \in \{\text{интересни,неинтересни}\}} P(v_j) P(a_1 = \text{"our"} | v_j)$$

$$P(a_2 = \text{"approach"} | v_j) \ldots (a_{111} = \text{"trouble"} | v_j)$$

Използваното в наивния Бейсов класификатор предположение за условната независимост на атрибутите в нашия контекст означава, че вероятностите на думите, намиращи се на определени позиции, не зависят от думите, намиращи се на други позиции. Очевидно обаче това предположение *е невярно*.

Например, вероятността да открием на някоя позиция думата “learning” е по-голяма, ако преди нея се намира думата “machine”. Въпреки че посоченото базово предположение е невярно, на практика нямаме голям избор, тъй като без него броят на вероятностите, които трябва да бъдат изчислени, е изключително голям. За радост наивният Бейсов класификатор работи учудващо добре и в повечето случаи на очевидно нарушаване на предположението за условна независимост, както е показано в някои литературни източници.

За да изчислим получения израз, трябва да имаме приближения за стойностите на $P(v_j)$ и $P(a_i = w_k|v_j)$, като с w_k тук ще означаваме k-тата дума в речника на английския език. Първият терм може да бъде оценен лесно като пропорцията (относителния дял) на съответния клас в обучаващите данни: *P(интересни) = 0.3*; *P(неинтересни) = 0.7*. Оценката на условните вероятности (например $P(a_1 =$ *“our”*$|$*неинтересни)*) е по-сложна, тъй като изисква да се оцени по един такъв терм за всяка комбинация от текстова позиция, английска дума и съответната целева стойност.

В речника на английския език има около 50000 различни думи; работим с 2 стойности на целевия атрибут и 111 позиции в текста, подлежащ на класифициране. Следователно трябва да оценим *2*111*50000 ≈ 10 000 000* комбинации за обучаващите данни.

За радост е възможно да се направи още едно разумно предположение, значително намаляващо броя на вероятностите, които трябва да оценим. Ще предположим, че вероятността да срещнем някоя конкретна дума w_k (например “chocolate”) не зависи от конкретната позиция на думата в текста. Формално това означава, че предполагаме, че атрибутите са независими и равномерно разпределени при зададена целева класификация, т.е. че $P(a_i = w_k|v_j) = P(a_m = w_k|v_j)$ за всички $i, j, k, m.$

По този начин можем да оценим цялото множество от вероятности $P(a_1 = w_k|v_j)$, $P(a_2 = w_k|v_j)$, … чрез една единствена, независима от позицията вероятност $P(w_k|v_j)$. Това означава, че са необходими оценките на само $2*50000$ различни стойности за различните $P(w_k|v_j)$.

За да завършим проектирането на нашия алгоритъм за самообучение, трябва да изберем подходящ метод за оценяването (пресмятането) на необходимите вероятности. Ще използваме известната от литературата формула

$$P(w_k|v_j) = \frac{n_k+1}{n+|\text{Речник}|},$$

където n е общият брой позиции на думите във всички обучаващи примери, които имат стойност на целевия атрибут v_j; n_k е броят на срещанията на думата w_k сред тези n позиции; $|Речник|$ е общият брой на различните думи, намерени в обучаващите данни.

Резюме

За да изчисли най-вероятната класификация на всеки нов пример, оптималният Бейсов класификатор комбинира предсказанията на всички алтернативни хипотези, претеглени от техните апостериорни вероятности.

Наивният Бейсов класификатор е метод за Бейсово обучение, който е доказал своята приложимост за множество практически задачи. Той се нарича „наивен“, защото използва улесняващото предположение, че атрибутните стойности са условно независими при зададена класификация на примера. Когато това предположение е изпълнено, наивният Бейсов класификатор извежда MAP класификацията. Дори когато това предположение не е изпълнено, използваният класификационен метод остава много ефективен.

Системи, основани на знания - зимен семестър, 2022/2023 учебна година

Тема 16: Клъстеризация

Какво представлява клъстеризацията?

Клъстеризация е дейността по групиране на набор от обекти (вектори или точки), така че обектите от една и съща група (наричана клъстер) да са подобни помежду си и да се различават от обектите в останалите клъстери.

На пръв поглед терминологията на клъстера изглежда ясна и точна: *група от сходни обекти*.

Но клъстерите, открити от различни алгоритми, могат да се различават значително по своите свойства. Следователно, съществен е изборът на подходящ алгоритъм при разрешаването на конкретен проблем.

Моделите за клъстеризация се разделят и на „hard” и „soft”. При „hard” клъстеризацията всеки обект попада в точно един клъстер. При „soft” клъстеризацията всеки обект може да попадне в повече от един клъстер, като за всеки клъстер, в който попада, обектът притежава съответна степен на принадлежност.

Алгоритъм *k*-means

Сравнително прост алгоритъм. Стреми се да раздели обектите на *k* (*k* е дадено естествено число) клъстера по следния начин:

- По случаен начин се избират центровете на всички клъстери;
- Повтарят се следните стъпки:
  - всеки обект се асоциира с (причислява към) клъстера с най-близък център;
  - замества се всеки център на клъстер със средното на всички обекти, асоциирани с него.

Описание на алгоритъма

Приемаме, че данните са в двумерното пространство и образуват k клъстера.

По случаен (напълно произволен) начин избираме k обекта $m_1^{(1)}$, … ,$m_k^{(1)}$, които наричаме *средни*.

Разпределяне

Добавяме всяка точка (всеки обект) x_p към клъстера с/около най-близкото средно. Така получаваме текущото множество от клъстери:

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \ \forall j, 1 \leq j \leq k\}$$

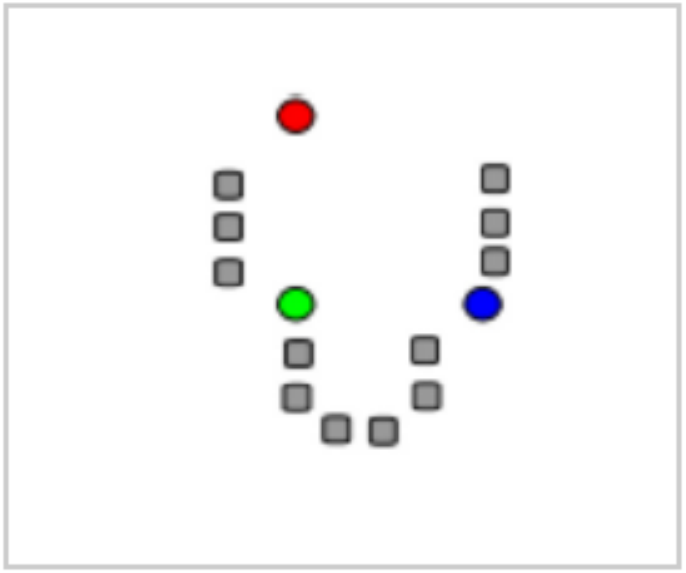
При това всяка точка x_p се асоциира с точно един клъстер $S^{(t)}$, дори и ако може да бъде асоциирана с два или повече клъстера.

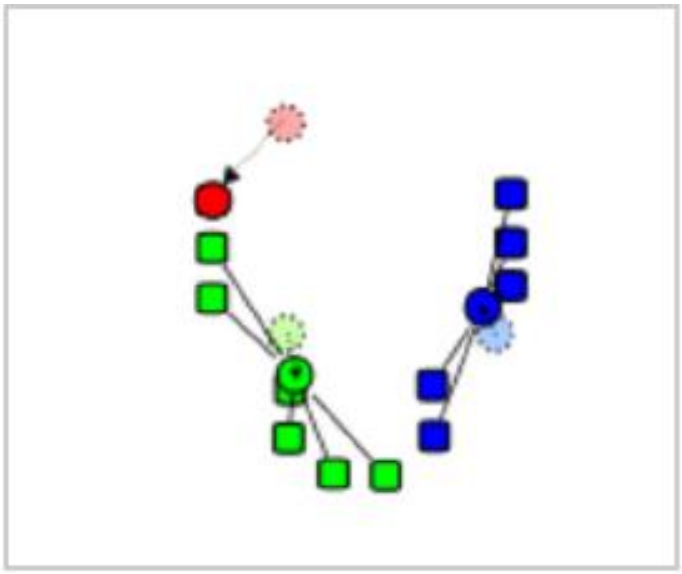
Обновяване

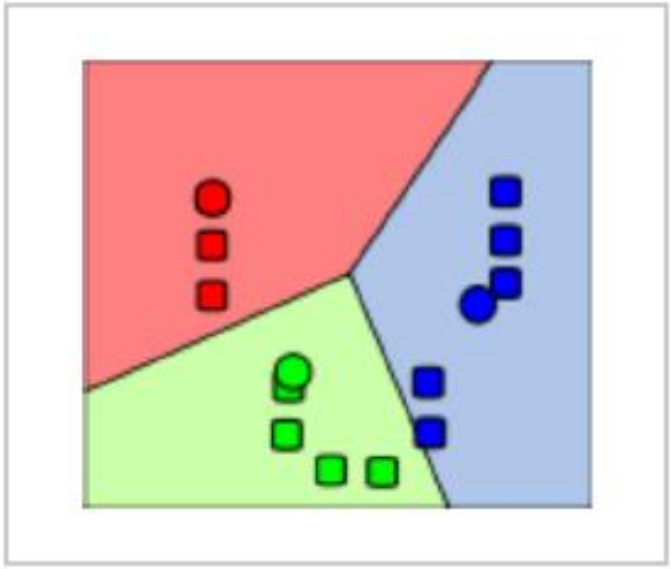
Обновяваме множеството от средните, като за нови средни избираме центроидите на новопостроените клъстери:

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

Пример







Оценка на алгоритъма

Алгоритмът *k*-means представлява техника за локално търсене с цел оптимизиране на разпръскването на данните.

Тъй като това е евристичен алгоритъм, няма гаранция, че ще се получи оптимално групиране. Резултатът може да зависи от избора на началните средни.

Алгоритъмът обикновено е много бърз, но има случаи, при които дори в двумерно пространство, може да отнеме експоненциално време $2^{\Omega(n)}$.

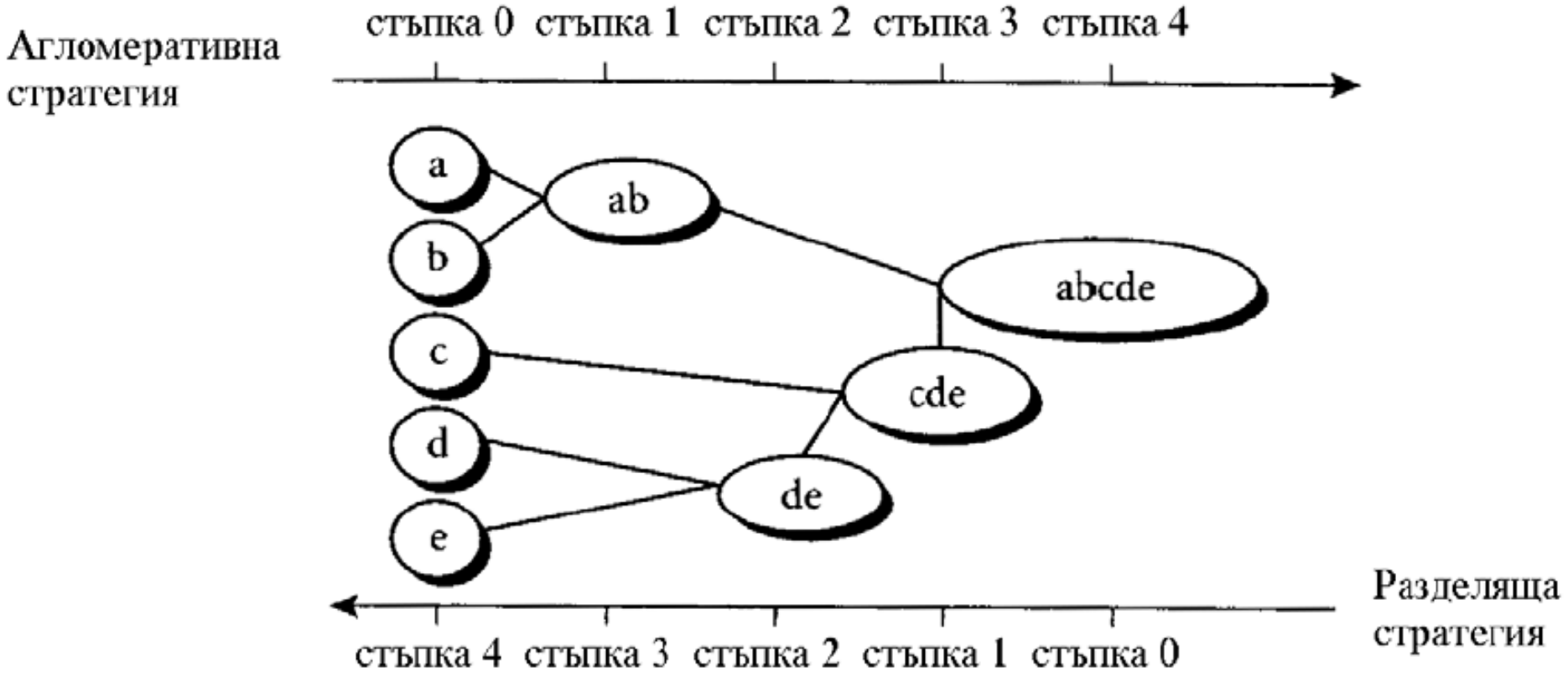
Такива случаи обаче рядко се срещат в практиката.

Йерархично клъстериране

Йерархичното клъстериране е метод, който създава йерархия от клъстери. За разлика от *k*-means, не е необходимо да зададем броя клъстери (*k*). Необходимо е само да посочим по какъв начин ще се сравняват обектите (например използваното разстояние). Като резултат получаваме дърво, листата на което са построените клъстери.

Има два типа йерархично клъстериране:

- *агломеративен* – подхожда „отдолу нагоре“, всеки обект стартира като собствен клъстер и с изкачването нагоре клъстерите се групират един с друг;
- *разделящ* – подхожда „отгоре надолу“, всички обекти стартират като един клъстер и със слизането надолу рекурсивно се разделят.


Агломеративна стратегия
стъпка 0 стъпка 1 стъпка 2 стъпка 3 стъпка 4
a
b
c
d
e
ab
de
cde
abcde
Разделяща стратегия
стъпка 4 стъпка 3 стъпка 2 стъпка 1 стъпка 0

Резултатът от прилагането и на двете стратегии за йерархично клъстериране се базира на избрания метод за определяне на *разстоянието между клъстери*.
Съществуват различни дефиниции на това разстояние, като всички се характеризират с определен начин за изчисляване на разстоянията между двойки обекти от различни клъстери.

Една от най-ранните и най-популярните мерки за разстояние между клъстери е *минималното разстояние*, което още се нарича *разстояние до най-близкия съсед.* Тя дефинира разстоянието между двата клъстера като разстояние между двойка(та) най-близки обекти от тези клъстери.

В такъв случай се говори за т. нар. *метод на единичното свързване* (*single-link clustering*).

Използването на тази мярка за разстояние между клъстери води до така наречения *„верижен ефект“*, при който дългите верижки от близо намиращи се точки (обекти) попадат в един и същ клъстер.

Методът на единичното свързване има едно важно свойство: ако две двойки клъстери се намират на едно и също разстояние помежду си, то няма никакво значение в какъв ред те ще се сливат или разделят – резултатът ще бъде един и същ.

Системи, основани на знания - зимен семестър,
2022/2023 учебна година

Тема 17:
Невронни мрежи

Същност на невронните мрежи

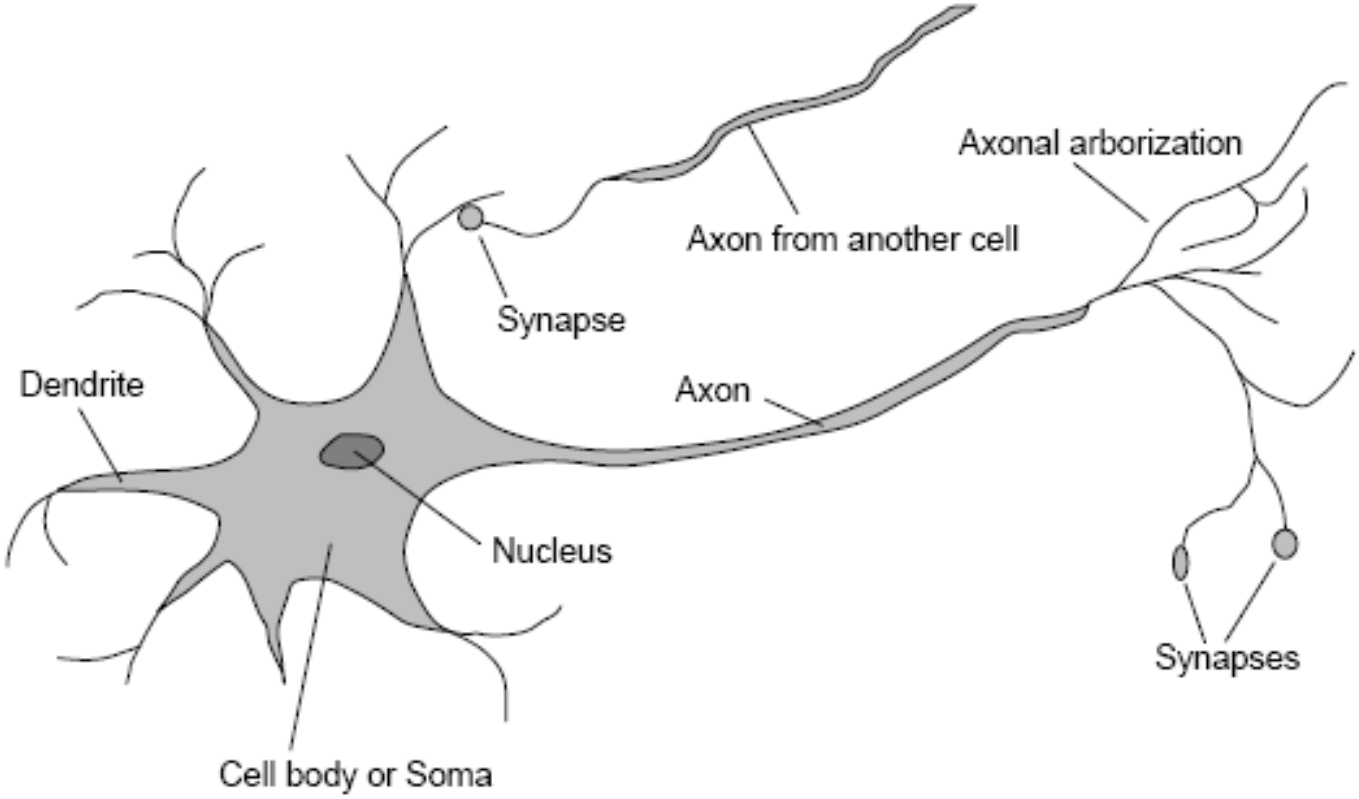
Невронните мрежи (НМ) и по-точно, изкуствените НМ са най-типичният и най-развитият представител на т. нар. *конекционистки подход* в Изкуствения интелект. Основните градивни елементи на (изкуствените) НМ са силно опростени модели на биологичните неврони, от които е съставен човешкият мозък.

Основни компоненти на биологичните неврони:

- *дендрити* – разклонени структури, които осигуряват сензорен вход към тялото на неврона (входни устройства на неврона);
- *тяло на неврона* – сумира мембранните потенциали между синапсите и дендритите и изпраща по аксона активиращо напрежение с определен размер;
- *аксон* – провежда електрическия сигнал от тялото на неврона до неговите синапси;

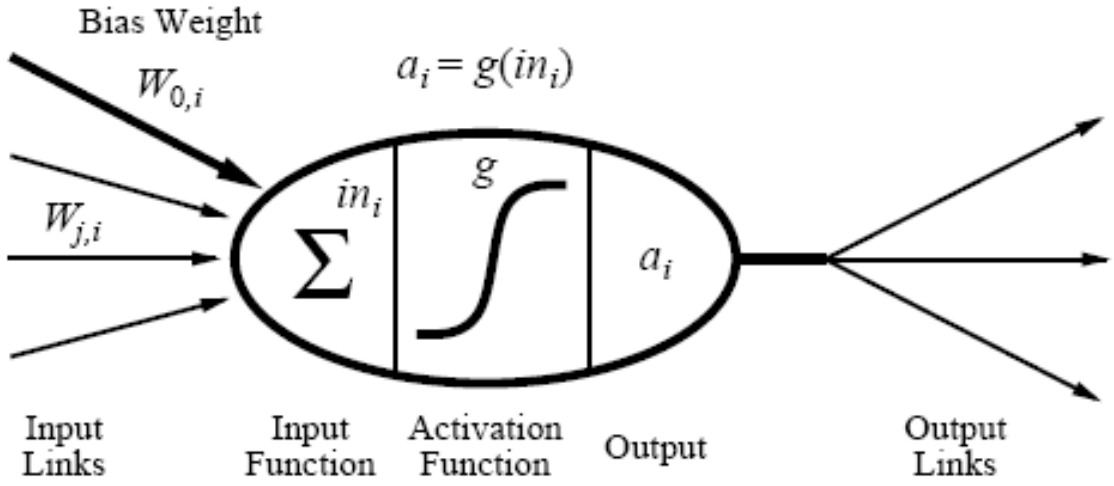
- *синапси* – трансформират активиращото напрежение, получено по аксона, в електрически сигнали (импулси), които възбуждат или подтискат активността на съседните неврони (изходни устройства на неврона).

Отделните неврони са свързани помежду си (дендритите на всеки неврон получават „входна информация“ от синапсите на други неврони и т.н.).


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Всяка изкуствена НМ е система от обработващи елементи (processing units), които са силно опростени модели на биологичните неврони и затова често условно се наричат (изкуствени) неврони. Обработващите елементи са свързани помежду си с връзки с определени тегла. Всеки обработващ елемент преобразува входните стойности (входните сигнали), които получава, и формира съответна изходна стойност (изходен сигнал), която разпространява към елементите, с които е свързан.

Това преобразуване се извършва на две стъпки. Най-напред се формира сумарният входен сигнал за дадения елемент, като за целта отделните входни стойности (сигнали) се умножават по теглата на връзките, по които се получават, и резултатите се събират. След това обработващият елемент използва специфичната за мрежата *активационна функция* (функция на изхода), която трансформира получения сумарен вход в изхода (изходния сигнал) на този елемент.


Bias Weight
$W_{0,i}$
$a_i = g(in_i)$
$W_{j,i}$
in_i
Σ
g
a_i
Input Links
Input Function
Activation Function
Output
Output Links

Основни типове модели на невронни мрежи

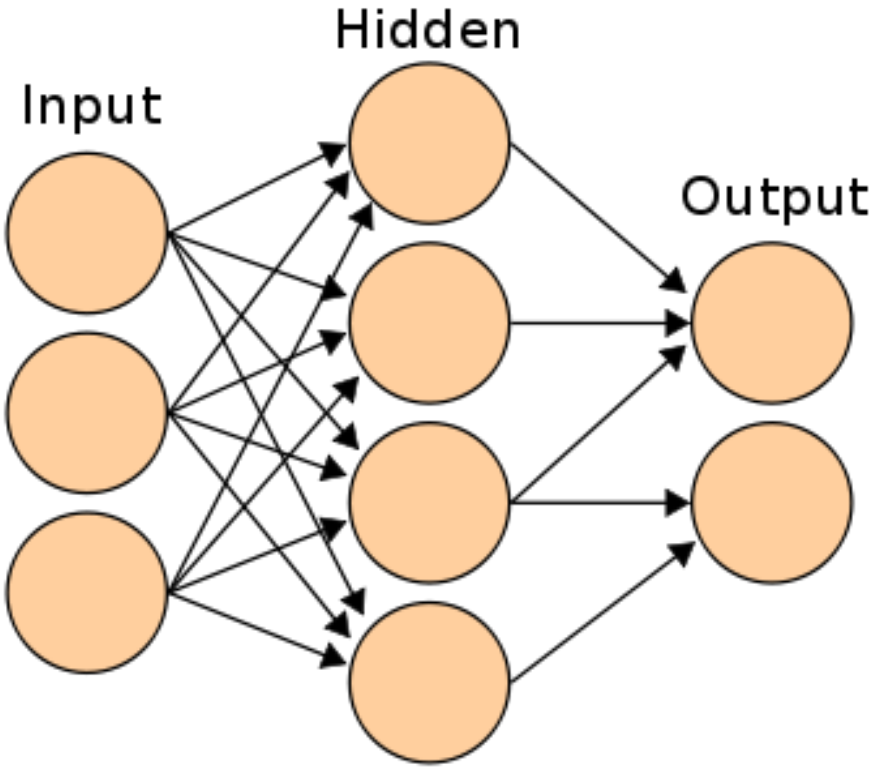
Специалистите определят пет основни характеристики на НМ, които обикновено се използват за класификационни признаци при определянето на типовете модели на НМ: *топологията на мрежата*, *параметрите на обработващите елементи*, *типът на входните и изходните стойности*, *методът за обучение на мрежата* и *използваното обучаващо правило*.

Според топологията на мрежата

Най-често срещани са НМ, съставени от няколко обособени (последователни) слоя от елементи, при които елементите от най-ниския слой играят ролята на входни устройства на мрежата (те възприемат сигнали от външната среда), а елементите от най-горния слой играят ролята на изходни устройства на мрежата (те извеждат резултата от работата на мрежата, който се получава на базата на входните сигнали и теглата на връзките между елементите).

Обикновено при тези НМ връзките са еднопосочни и свързват елементите от един слой с елементи на слоя, разположен непосредствено над него.

В зависимост от броя на слоевете в мрежата се говори за *двуслойни НМ* (при тях има само един входен и един изходен слой и липсват т. нар. *вътрешни* или *скрити слоеве*) и *многослойни НМ* (при тях има поне един скрит слой).



Многослойна (трислойна) НМ

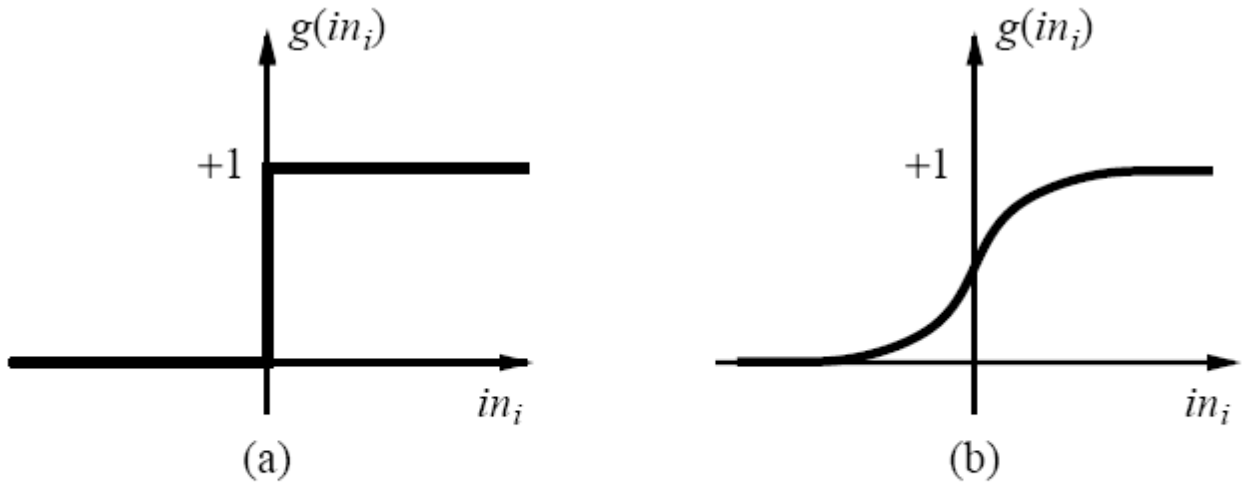
Според параметрите на елементите

Тук от значение са посоката и теглата на връзките, видът на активационната функция и др.

Както беше посочено, при повечето НМ връзките са еднопосочни и са ориентирани от елементите от даден слой към елементите от слоя, разположен непосредствено над него (това са т. нар. *прави мрежи*). Съществуват обаче и т. нар. *рекурентни мрежи* (такива са например мрежите на Хопфийлд), при които всеки елемент е свързан с двупосочни връзки с всички свои съседи. При тези мрежи понятието слой до голяма степен губи своя смисъл.

По отношение на теглата на връзките се говори за т. нар. *възбуждащи връзки* (връзки с положителни тегла) и *подтискащи връзки* (връзки с отрицателни тегла).

Най-често срещаните типове активационни функции са *стъпаловидна функция* (при персептрона) и *сигмоид* (при НМ с обратно разпространение на грешката).



(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Според типа на входните и изходните стойности

Входните стойности на мрежата (т.е. сигналите, които получават елементите от входния слой) могат да бъдат *двоични* (0 или 1) или *аналогови* (реални числа). Същото се отнася и за изходните стойности на мрежата (стойностите/сигналите, които елементите от изходния слой извеждат към „околната среда“). В случай, че изходните стойности на мрежата трябва да бъдат двоични, се налагат допълнителни изисквания към вида на активационната функция.

Според обучаващото правило

Най-често НМ се използват за решаване на задачи, свързани с разпознаване (класификация). При това създаването на НМ, предназначена за решаване на дадена конкретна задача, обикновено се извършва по следната обща схема. Най-напред се определя топологията на мрежата, т.е. броят на слоевете и броят на елементите във всеки слой. Броят на елементите от входния слой се определя от размерността на входните данни, а броят на елементите от изходния слой – от броя на разпознаваните класове. Броят и размерите на скритите слоеве се определят итеративно в зависимост от предметната област и конкретната задача.

След определянето на топологията на мрежата се преминава към нейното обучаване, т.е. към определянето на подходящи стойности на теглата на връзките между елементите. За целта най-често отначало се задават случайни стойности на търсените тегла, след което те итеративно се променят (уточняват) по такъв начин, че на следващата стъпка новият изход на мрежата да бъде по-добър от стария. Правилото, по което се променят теглата на връзките, се нарича *обучаващо правило* (обучаващ алгоритъм) на мрежата.

Примери за по-известни обучаващи правила: правило за обучение на персептрона (алгоритъм за обучение с фиксирано нарастване), алгоритъм за обучение с обратно разпространение на грешката, правило на състезателното обучение.

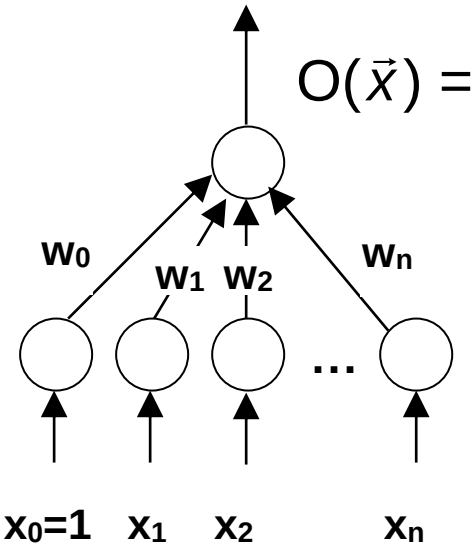
Според метода на обучение

Съществуват два основни типа обучение на НМ – *обучение с учител* и *обучение без учител*. При обучението с учител се следи разликата между получения и очаквания изход на мрежата (учителят задава очаквания изход на мрежата) и итеративно се извършват корекции на теглата на връзките съгласно избраното обучаващо правило. При обучението без учител липсват данни за правилния изход на мрежата. Теглата на връзките се настройват така, че представянето на данните в мрежата да е най-добро съгласно използвания (зададения) критерий за качеството на представянето.

Популярни модели на невронни мрежи и алгоритми за обучение

Персептрон

Персептронът (по-точно, двуслойният персептрон) е двуслойна НМ с един елемент в изходния слой, който често се нарича сумиращ процесор на персептрона.



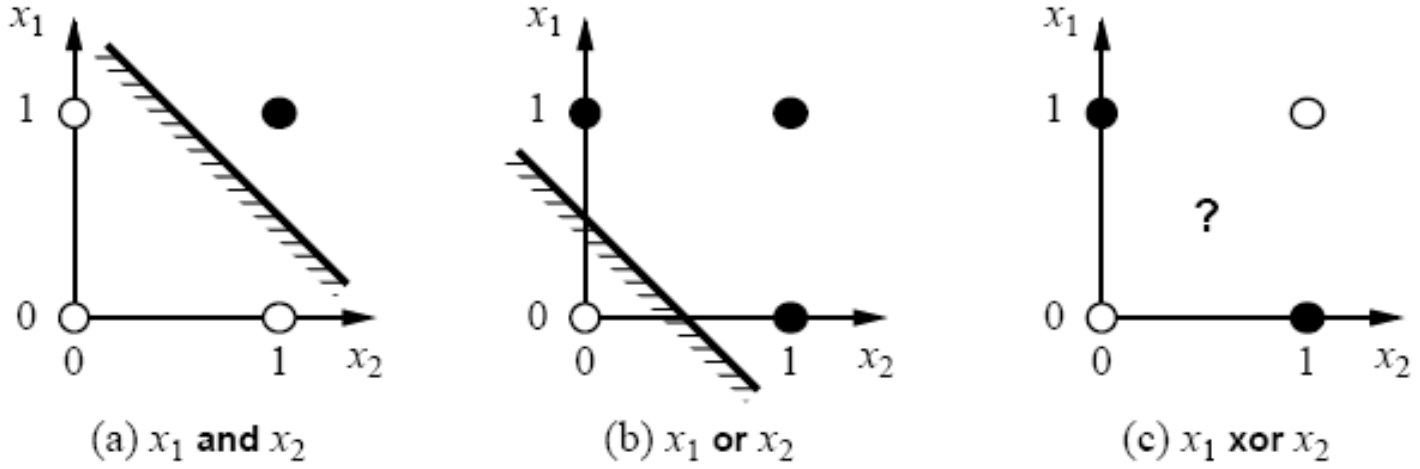
$$O(\vec{x}) = \begin{cases} 1, \text{ ако } g(\vec{x})>0 \\ 0, \text{ ako } g(\vec{x})<0 \\ \text{Резултатът е неопределен, ако } g(\vec{x})=0 \end{cases}$$

Тук $g(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i = \vec{w}\vec{x}$ ($\vec{w}$ - теглови вектор,

$\vec{x}$ - входен вектор)

Резултатът от работата на персептрона е решаването на задача за разпознаване при дадени два класа, разделими с хиперравнина (в двумерния случай - разделими с права линия, т.е. линейно разделими).

Примери



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Алгоритъм за обучение на персептрона – т. нар. *алгоритъм за обучение с фиксирано нарастване*. Доказана е теорема за сходимост, според която ако съществува множество от тегла, определящи хиперравнина, която разделя двата класа, то с помощта на този алгоритъм могат да бъдат намерени подходящи стойности на търсените тегла на връзките между елементите от входния и изходния слой.

Формулировка на алгоритъма за обучение на персептрона

Дадено: множество от обучаващи примери за задача за разпознаване (класификация) с n входни характеристики (x_1, … , x_n) и два изходни класа.

Търси се: множество от тегла (w_0, w_1, … , w_n), които са такива, че персептронът дава стойност 1 точно тогава, когато входните данни съответстват на обект, принадлежащ на първия клас.

Изчислителна схема:

1. Създава се персептрон с $n+1$ входни елемента и $n+1$ тегла, където допълнителният входен елемент x_0 винаги има стойност 1.
2. Инициализират се със случайни реални стойности теглата (w_0, w_1, … , w_n).
3. При текущите стойности на теглата w_i се класифицират примерите от обучаващото множество, след което от тях се подбират само тези, които са класифицирани неправилно.
4. Ако всички обучаващи примери са класифицирани правилно, като резултат от работата на алгоритъма се извеждат текущите стойности на теглата w_i и ***край***.

5. В противен случай се пресмята векторът $\vec{s}$ като сума на неправилно класифицираните обучаващи вектори $\vec{x}$. При формирането на сумата $\vec{s}$ в нея с положителен знак участват всички вектори $\vec{x}$, за които персептронът неправилно е дал резултат 0, а с отрицателен знак – тези, за които персептронът неправилно е дал резултат 1. Така полученият вектор $\vec{s}$ се умножава с предварително избран скаларен коефициент η. Стойността на η ($0<\eta<1$) определя скоростта на сходимост на алгоритъма и изборът ѝ зависи от спецификата на решаваната задача.

6. Модифицират се теглата (w_0, w_1, … , w_n), като към тях се прибавят съответните елементи на вектора $\vec{s}$. Преминава се към т. 3.

Забележка. Нека е дадено множество от обекти, които подлежат на разпознаване/класификация. Предполага се, че всеки обект се характеризира с набор от съществени признаци. На базата на използваните признаци се определя *признаковото пространство* (*пространството на признаците*). Всеки обект е точка в това пространство. Тогава задачата за дефиниране на класовете се свежда до това за всеки клас C_i да се намери функция $g_i(\vec{x})$, която е такава, че $\vec{x} \in C_i$ тогава и само тогава, когато $g_i(\vec{x}) > g_j(\vec{x})$ за всяко $j \neq i$.

При известни функции $\{g_i\}$ задачата за класификацията на даден обект $\vec{x}$ се свежда до намиране на това k, за което $g_k(\vec{x})>g_j(\vec{x})$ за всяко $j\neq k$ (тогава $\vec{x}\in C_k$). При наличие само на два класа горната задача се свежда до намиране или използване на т. нар. *разделящо правило* (*разделяща функция*) g със свойството: $\vec{x}\in C_1$, когато $g(\vec{x})>0$, и $\vec{x}\in C_2$, когато $g(\vec{x})<0$. В този случай точките, за които $g(\vec{x})=0$, образуват границата между класовете.

Невронни мрежи с обратно разпространение на грешката

Това са НМ с поне един скрит слой, при които съществуват връзки от всеки елемент от даден слой към всеки елемент от непосредствено следващия слой.

Активационната функция на този тип мрежи има вида $output = \frac{1}{1+e^{-sum}}$, където *sum* е сумата от всички входни стойности, умножени с теглата на съответните връзки.

Алгоритъм за обучение с обратно разпространение на грешката: включва множество *епохи*, в рамките на всяка от които последователно се разглеждат всички примери от обучаващото множество. За всеки обучаващ пример се извършват т. нар. *прав пас* (разпространение на активационните стойности) и *обратен пас* (анализ на грешките на изхода и разпространение на корекциите на теглата на връзките между елементите от различните слоеве).

Следва описание на алгоритъма за случая на трислойна невронна мрежа (НМ с точно един скрит слой).

Дадено: множество от обучаващи примери (наредени двойки $(\vec{x},\vec{y})$ от входни и съответни изходни стойности на мрежата).

Търсят се: подходящи стойности $\overrightarrow{w_1}$ и $\overrightarrow{w_2}$ на теглата на връзките в трислойна НМ, която по дадени входни стойности от обучаващото множество връща съответните изходни стойности от същото множество (т.е. работи коректно върху даденото множество от обучаващи примери).

Изчислителна схема:

1. Нека A е броят на елементите от входния слой (съответен на дължината на входните вектори от обучаващото множество) и C е броят на елементите от изходния слой (съответен на дължината на изходните вектори от обучаващото множество). Избира се подходяща стойност на B, равна на броя на елементите от скрития слой. Има многобройни резултати, които могат да се използват при експериментиране в процеса на определянето на B. Въвеждаме следните означения:
   x_i – стойностите (активационните равнища) на елементите от входния слой (при това се полага $x_0 = 1$);
   h_i – стойностите (активационните равнища) на елементите от скрития слой (при това се полага $h_0 = 1$);

o_i – стойностите (активационните равнища) на елементите от изходния слой;
$w_1(i,j)$ – теглата на връзките между елементите от входния и скрития слой (i – индекс на елемента от входния слой, j – индекс на елемента от скрития слой);
$w_2(i,j)$ – теглата на връзките между елементите от скрития и изходния слой (i – индекс на елемента от скрития слой, j – индекс на елемента от изходния слой).

2. Инициализират се теглата на връзките в мрежата. Всяко тегло получава случайна стойност в интервала (-0.1,0.1).
3. Инициализират се стойностите (активационните равнища) на допълнителните елементи: x_0 = 1.0, h_0 = 1.0. Тези стойности не се променят на следващите стъпки от изпълнението на алгоритъма.
4. Избира се обучаващ пример, т.е. двойка ($\vec{x}$,$\vec{y}$) от входен и съответен изходен вектор от обучаващото множество.

5. Разпространяват се активационните стойности от входния към скрития слой, като за целта се използва активационната функция на мрежата:
   $h_j = \frac{1}{1+\exp(-\sum_{i=0}^{A} w_1(i,j)x_i)}$ за j= 1,2, ... ,B.
6. Разпространяват се активационните стойности от скрития към изходния слой:
   $o_j = \frac{1}{1+\exp(-\sum_{i=0}^{B} w_2(i,j)h_i)}$ за j= 1,2, ... ,C.
7. Пресмятат се грешките, получени при елементите от изходния слой. Нека означим тези грешки с $\delta_2(j)$. Всяка от тях се пресмята с помощта на реалния изход на мрежата o_j и правилния изход от обучаващия пример y_j:
   $\delta_2(j) = o_j(1 - o_j)(y_j - o_j)$ за j = 1,2, … ,C.

8. Пресмятат се грешките на елементите от скрития слой.Нека означим тези грешки с $\delta_1(j)$:
   $\delta_1(j) = h_j(1 - h_j)\sum_{i=1}^{C} \delta_2(i) w_2(j,i)$ за $j = 1,2, \ldots, B$.
9. Уточняват се стойностите на теглата на връзките между елементите от скрития и изходния слой:
   $w_2(i,j) = w_2(i,j) + \Delta w_2(i,j)$,
   $\Delta w_2(i,j) = \eta \delta_2(j) h_i$ за $i = 0, \ldots, B$ и $j = 1, \ldots, C$.
   Тук коефициентът η има същия смисъл, както и при алгоритъма за обучение на персептрона.
10. Уточняват се стойностите на теглата на връзките между елементите от входния и скрития слой:
    $w_1(i,j) = w_1(i,j) + \Delta w_1(i,j)$,
    $\Delta w_1(i,j) = \eta \delta_1(j) x_i$ за $i = 0, \ldots, A$ и $j = 1, \ldots, B$.

11. Преминава се към стъпка 4 и се повтарят действията от стъпки 4 – 10 за всички останали обучаващи примери.

Така завършва една *епоха* от работата на алгоритъма. Повтарят се толкова епохи, колкото е необходимо за достигане на коректен изход на мрежата за всички обучаващи примери.

Алгоритъмът за обучение с обратно разпространение на грешката има *ниска скорост на сходимост*, която води до ниска скорост на обучение на невронната мрежа и необходимост от *голям брой обучаващи примери*.

НМ с обратно разпространение на грешката се използват широко поради голямата им мощност.

Обучение на невронни мрежи без учител (самообучение на невронни мрежи)

При обучението без учител (т.е. при самообучението) се предполага, че не съществува обратна връзка със средата, в която работи мрежата, т.е. няма кой да определи какъв би трябвало да бъде изходът на мрежата и дали формираният от нея изход е коректен. Мрежата трябва сама да открие образци, характеристики, регулярности, корелации или категории във входните данни и да ги кодира на изхода. Затова изкуствените неврони и връзките между тях трябва да притежават (да демонстрират) някаква степен на ***самоорганизация***.

В резултат на самообучението може да се получи нещо полезно само когато съществува ***изобилие от входни данни***. Без изобилие не може да се намерят някакви характеристики на данните, тъй като в тях има и случайни шумове. В този случай ***изобилието дава знание***.

Типът на образеца, който самообучаващата се мрежа открива във входните данни, зависи от ***архитектурата ѝ***. Обикновено самообучаващите се мрежи имат проста архитектура – най-често те са прави и включват един входен и един изходен слой. При това обикновено изходите са много по-малко от входовете.

Някои възможни резултати от работата на самообучаваща се невронна мрежа:

Клъстеризация. Множество от двоични изходи, само един от които е активен в определен момент от времето, могат да определят към коя от няколко категории принадлежи входният образец. Всеки клъстер от подобни или близки образци ще бъде класифициран като един изходен клас.

Кодиране. Изходът може да бъде кодирана версия на входа в по-малко битове, поддържайки толкова необходима информация, колкото е възможно. Това се използва за компресиране на данни преди подаването им по канал с ограничен обхват, предполагайки, че може да се конструира и обратна декодираща мрежа.

В повечето случаи архитектурите и обучаващите правила се основават на интуитивни предположения. В някои случаи се използват и оптимизационни подходи (цели се максимална икономичност на представянето или максимизиране на някакво количество, например на съдържанието на информацията, или минимизиране на изменението на изхода).

Един от най-често използваните методи (подходи) за самообучение е т. нар. ***състезателно обучение***. При него ***само един изходен неврон или само един неврон от определена група е активен***, т.е. има ненулева активност (активационно ниво, стойност) в даден момент от времето. Изходните неврони се състезават да бъдат активни и затова често се наричат ***„печелившият-взема-всичко“*** неврони или ***„изпреварващи-всички“*** елементи (възли).

Целта и тук е да се клъстеризират или категоризират входните данни. Подобни входове би трябвало да бъдат класифицирани в една и съща категория и затова би трябвало да активират един и същ изходен неврон. Класовете (клъстерите) трябва да бъдат открити от самата мрежа на основата на корелации между входните данни.

Стандартно състезателно обучение

Мрежите, които използват стандартното състезателно обучение, имат ***един входен и един изходен слой***. Всеки от изходните възли O_i е свързан с всеки от входните възли ξ_j чрез възбуждаща връзка с тегло $w_{ij} \geq 0$. Ще разглеждаме само мрежи с ***бинарен вход и изход*** (с входни и изходни стойности от множеството {0,1}). Само един от изходните неврони, наречен ***победител***, може да бъде активен (да има ненулева стойност) в даден момент. Обикновено това е възелът с най-голяма стойност (активност)

$h_i = \sum_j w_{ij}\xi_j = \overrightarrow{w_i}\overrightarrow{\xi}$ за дадения входен вектор $\overrightarrow{\xi}$.

Затова неравенството

(1) $\overrightarrow{w_{i*}}\overrightarrow{\xi} \geq \overrightarrow{w_i}\overrightarrow{\xi}$ за всяко *i*

дефинира побеждаващия неврон с $O_{i*} = 1$.

Като правило се изисква теглата за всеки изходен възел да са нормирани:

$|\overrightarrow{w_i}| = 1$ за всяко *i*.

В такъв случай за даден входен вектор $\vec{\xi}$ побеждаващият изходен неврон *i** с O_{i*} = 1 се дефинира посредством неравенството

(2) $|\overrightarrow{w_{i*}}-\vec{\xi}| \leq |\overrightarrow{w_i}-\vec{\xi}|$ за всяко *i*.

Една мрежа от тип „печелившият-взема-всичко“ реализира класификатор на образци, като използва критерия (1) или (2). ***Задачата за обучението ѝ*** е свързана с ***намиране на тегловите вектори*** $\overrightarrow{w_i}$ ***при изискването мрежата да намира по подходящ начин клъстери във входните данни***.

Няма принципно значение начинът, по който се реализира мрежа от тип „печелившият-взема-всичко“. При компютърна симулация винаги може да се осъществи търсене на максималното h_i. Често срещана (и по-естествена в определен смисъл) е и ситуацията, при която изходните неврони се състезават помежду си с цел излъчване на победител чрез ***странично подтискане***: всеки неврон подтиска другите чрез връзки с отрицателни тегла и евентуално се самовъзбужда чрез връзка с положително тегло. За целта страничните тегла и нелинейната активационна функция трябва да бъдат подбрани коректно – така, че да е сигурно, че ще бъде избран точно един изход и колебанията ще бъдат избегнати.

В такъв случай най-общо алгоритъмът на стандартното състезателно обучение изглежда по следния начин:

1. Присвояват се малки случайни стойности на теглата. Важно в този момент е да няма симетрия (т.е. теглата да са напълно различни).
2. Избира се входен вектор от обучаващото множество.
3. Пресмята се началната стойност (активност, активационно ниво) за всеки от изходните неврони.
4. Изходните неврони се състезават, докато само един от тях остане активен.

5. Увеличават се теглата на връзките между активния изходен неврон и активните входни неврони и се намаляват теглата на връзките между активния изходен неврон и неактивните входни неврони така, че векторът от тези тегла да остане нормиран.
6. Стъпки 2 – 5 се повтарят за всички входни вектори за много епохи.