Relevance.

0. Úvod.

Je-li strojové učení (přesněji řízené induktivní učení z příkladů) využíváno při řešení složitého a rozsáhlého (co do objemu dat) problému, objevuje se potřeba zaměřit se na (nejvíce) relevantní informace. Jde o

- relevantní atributy pro reprezentaci dat
- relevantní příklady pro řízení procesu učení.

V tomto textu se budeme zabývat převážně prvním tématem (které bylo také daleko intenzivněji studováno a zpracováno v literatuře), druhé pojednáme stručněji. Při jeho přípravě jsme vycházeli především z přehledového článku [3], v menší míře i z [19]. Součástí textu je též (poměrně obšírný) výběr literatury, ze které je možno čerpat podrobnější informace; uvedené citace (a ještě řada dalších) pocházejí většinou také z těchto zdrojů.

Nyní se pokusíme stručně vymezit prostředí, ve kterém se budeme dále pohybovat. Předpokládáme, že objekty našeho zájmu jsou popsány pomocí n atributů $a_1, \ldots, a_n$ s množinami hodnot $V_1, \ldots, V_n$. Atributy mohou být booleovské (červený ?), nominální (jaká barva ?) nebo spojité (jaká vlnová délka ?). Je-li pro daný objekt hodnota atributu a_i rovna x_i, je tento objekt charakterizován n-ticí $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$, které budeme říkat instance; množina $X = V_1 \times \ldots \times V_n$ je pak prostor (možných) instancí.

Na obecné úrovni se učení dá velmi často chápat jako učení se funkci. Předpokládejme tedy, že existuje nějaká funkce f (často se pro ni užívá termínu cílová funkce), která každé instanci přiřazuje nějaké ohodnocení (často se též říká klasifikaci), které opět může být booleovské, nominální nebo spojité. Dvojici $\langle \boldsymbol{x}, f(\boldsymbol{x}) \rangle$, kde $\boldsymbol{x} \in X$, budeme říkat příklad. Funkci f ovšem (celou) neznáme; k dispozici máme pouze její tzv. vzorek (množinu tréninkových příkladů nebo také tréninkovou množinu) S, tedy množinu pouze některých (relativně málo) dvojic $\langle$ ***instance,*** *ohodnocení*(***instance***) $\rangle$. Pro další úvahy je rozumné předpokládat, že na množině X instancí máme dánu distribuci (rozložení pravděpodobnosti) D a že vzorek S vznikl opakovaným náhodným výběrem z X v souladu s D. Cílem učení je pak na základě znalosti tréninkové množiny S vytvořit (tj. definovat na X) funkci (často nazývanou hypotéza) h, která by „dobře aproximovala“ cílovou funkci f (myslí se dobře aproximovala na celém univerzu X). Pod těmito slovy se obvykle rozumí jednak to, že h je konzistentní se vzorkem, tj. že pro instance $\boldsymbol{x} \in S$ se hodnoty $f(\boldsymbol{x})$ a $h(\boldsymbol{x})$ shodují, jednak že je korektní, tedy že chyba hypotézy h je malá (nejlépe nulová). Chybou h přitom rozumíme pravděpodobnost, že pro instanci $\boldsymbol{x}$ nepatřící do S a vybranou z X náhodně v souladu s D, se hodnoty $f(\boldsymbol{x})$ a $h(\boldsymbol{x})$ liší. Úspěšný učící algoritmus je tedy takový, který v tomto smyslu „dobře extrapoluje“ cílovou funkci i mimo tréninkovou množinu S. Otázky, které se v této souvislosti vynořují (například jak se vůbec dá posuzovat korektnost hypotézy h, když při jejím vytváření máme k dispozici pouze tréninkovou množinu S), spadají do oblasti tzv. PAC - učení (Probably Approximately Correct Learning) – viz například [5] nebo [15].

Často se při výkladu učení z příkladů omezujeme na případ, kdy cílová funkce je dvouhodnotová. Takovéto omezení nepředstavuje žádnou podstatnou újmu na obecnosti. Představme si, že možné instance jsou geometrické útvary v trojrozměrném prostoru (popsané pomocí počtu stěn, jejich tvaru a velikosti, jejich úhlů atd.) a cílová funkce má hodnoty krychle, kvádr, hranol, jehlan , ... ; při učení se tedy (pomocí tréninkové množiny) učíme rozeznávat různé geometrické útvary. Stejného efektu ovšem dosáhneme, když uvedenou funkci nahradíme skupinou (dvouhodnotových) funkcí, z nichž jedna bude objektu (instanci) přiřazovat hodnotu 1, je-li to krychle (a jinak 0), druhá mu přiřadí 1 právě když jde o kvádr atd. a budeme se učit všem takovým funkcím. V případě dvouhodnotové cílové funkce (někdy se místo toho říká cílového predikátu) se obvykle mluví o učení pojmu (concept learning) a cílová funkce (predikát) se značí c. V našem textu se tohoto případu ve shodě s použitou literaturou přidržíme (i když v podstatě pouze pro větší názornost doprovodných příkladů a představ). Poznamenejme ještě, že cílová funkce c může být deterministická nebo pravděpodobnostní; ve druhém případě není pro instanci $\boldsymbol{x}$ hodnota $c(\boldsymbol{x})$ jedno určité ohodnocení, nýbrž pravděpodobnostní rozložení na množině možných ohodnocení.

1. Výběr relevantních atributů.

Na obecné úrovni můžeme proces učení pojmu rozložit na dvě části , totiž rozhodnout, které atributy použijeme pro popis pojmu a rozhodnout, jak je budeme kombinovat. Z tohoto hlediska je výběr relevantních (a eliminace irelevantních) atributů klíčový. Na praktické úrovni chceme vytvářet induktivní algoritmy, které

pracují dobře v oblastech s mnoha irelevantními atributy. Konkrétněji řečeno chceme, aby počet tréninkových příkladů, potřebných k dosažení požadované úrovně přesnosti (tzv. složitost vzorku v PAC-učení), rostl pomalu s počtem přítomných (použitých) atributů.

Induktivní algoritmy učení se mohou podstatně lišit v tom, do jaké míry kladou důraz na relevanci atributů. Jedním extrémem je učení jednoduchou metodou nejbližšího souseda (nearest neighbor learning algorithm), což je základní metoda tzv. případového učení (instance-based learning). Při ní se vlastní učení redukuje na ukládání tréninkových příkladů do paměti. Testovací (dotazované) instanci pak přiřadí hodnotu stejnou, jako je hodnota „nejbližšího" uloženého tréninkového příkladu; pro výpočet vzdálenosti přitom používá všech atributů, které má k dispozici. I když má tato metoda vynikající asymptotickou přesnost (viz [8]), je zřejmé, že irelevantní atributy učení výrazně zpomalují. Analýza průměrného případu této metody provedená v [21] ukazuje, že počet tréninkových příkladů, potřebných k dosažení požadované přesnosti, roste exponenciálně s počtem irelevantních atributů (dokonce již pro tak jednoduché cílové pojmy, jako jsou pojmy vyjádřitelné elementární konjunkcí atributů). Na druhém konci této pomyslné škály jsou systémy, které se pokouší explicitně vybrat relevantní atributy a vyřadit irelevantní. Nejjednodušším příkladem těchto snah jsou metody učení logických popisů; existují samozřejmě i důmyslnější metody výběru relevantních atributů, které mohou zlepšit libovolnou induktivní metodu (včetně metody nejbližšího souseda). Teoretické výsledky týkající se takovýchto metod jsou velmi slibné. Například v [4] se ukazuje, že pokud platí, že omezí-li se učící algoritmus na „malou" podmnožinu atributů, může významně redukovat počet hypotéz, připadajících v úvahu pro učení, pak existuje také odpovídající redukce složitosti vzorku, potřebného pro zaručení požadované přesnosti hypotézy. Někde mezi těmito případy se nacházejí metody používající vážení atributů, které nevybírají explicitně podmnožiny atributů ale přesto se snaží dosáhnout dobrého (ve smyslu zmenšení závislosti na irelevantních atributech) chování učícího algoritmu.

1.1. Různé definice relevance.

V literatuře o strojovém učení se (v souvislosti s řízeným induktivním učením z příkladů) objevuje řada definic relevance (relevantní vůči čemu ?). Uvedeme (podle [3]) a okomentujeme nyní některé z nich a připojíme několik obecných poznámek o algoritmech pro výběr atributů.

Definice 1 (relevance vzhledem k cíli). Atribut a_i je relevantní cílovému pojmu (cílové funkci) c, jestliže existuje dvojice instancí $\boldsymbol{x}$ a $\boldsymbol{y}$ v prostoru X instancí taková, že $\boldsymbol{x}$ a $\boldsymbol{y}$ se liší pouze hodnotou x_i atributu a_i a přitom $c(\boldsymbol{x}) \neq c(\boldsymbol{y})$.

Jinými slovy existuje instance taková, že změna hodnoty x_i atributu a_i má vliv na klasifikaci určenou cílovou funkcí. Učící algoritmus má ovšem přístup pouze ke vzorku S tréninkových příkladů a nemusí tedy být schopen rozhodnout, zda atribut a_i je či není v uvedeném smyslu relevantní. Tato definice se často užívá při teoretické analýze učících algoritmů, kde je pojem relevance využit při důkazu tvrzení o konvergenci algoritmu, spíše než v samotném algoritmu.

Definice 2 (silná relevance vzhledem ke vzorku resp. rozložení pravděpodobnosti). Atribut a_i je silně relevantní vzorku S, jestliže existuje dvojice příkladů $\boldsymbol{s}_1$ a $\boldsymbol{s}_2$ z S taková, že jim příslušné instance se liší pouze hodnotou x_i atributu a_i a přitom mají různou klasifikaci. Atribut a_i je relevantní cílovému pojmu c a rozložení pravděpodobnosti D, jestliže existují instance $\boldsymbol{x}$ a $\boldsymbol{y}$ mající nenulovou pravděpodobnost při rozložení D takové, že $\boldsymbol{x}$ a $\boldsymbol{y}$ se liší pouze hodnotou x_i atributu a_i a přitom $c(\boldsymbol{x}) \neq c(\boldsymbol{y})$.

Definice 3 (slabá relevance vzhledem ke vzorku resp. rozložení pravděpodobnosti). Atribut a_i je slabě relevantní vzorku S resp. cílovému pojmu c a rozložení pravděpodobnosti D, jestliže existuje množina atributů, po jejímž odstranění bude v příslušném smyslu silně relevantní .

Tyto (dva) pojmy relevance jsou užitečné z hlediska učícího algoritmu, který chce rozhodnout, které atributy si má podržet a které ignorovat. Silně relevantní atributy je vždy důležité zachovat (alespoň v tom smyslu, že jejich odstranění zanáší do vzorku dvojznačnost); slabě relevantní atributy mohou či nemusí být důležité podle toho, které (jiné) atributy jsou ignorovány.

Z jiného pohledu se na relevanci můžeme dívat jako na míru složitosti. Místo toho, abychom zjišťovali, které atributy jsou relevantní (tedy aby algoritmus explicitně vybíral podmnožinu atributů), chceme pomocí relevance říci, jak „složitá" je cílová funkce (a chceme, aby algoritmus pracoval dobře, je-li tato hodnota malá).

Definice 4 (relevance jakožto míra složitosti). Pro daný vzorek S tréninkových příkladů a třídu C cílových pojmů definujeme $r(S, C)$ jako počet atributů relevantních podle definice 1 takovému pojmu z C, který má mezi pojmy s nejmenší chybou na S nejmenší počet relevantních atributů.

Jinak řečeno, ptáme se po nejmenším počtu atributů potřebných pro dosažení optimálního chování pojmů z C na S. Pro zvýšení robustnosti můžeme definici modifikovat tak, že připustíme pojmy z C s „téměř minimální" chybou na S, pokud jim přísluší menší množina relevantních atributů.

Uvedené definice relevance nezávisí na konkrétním algoritmu použitém pro učení. Není tedy zaručeno, že je-li atribut relevantní, bude také nutně užitečný (a naopak).

Definice 5 (inkrementální užitečnost). Pro daný vzorek S, učící algoritmus L a množinu atributů A, atribut a_i je inkrementálně užitečný pro L vzhledem k A, jestliže přesnost hypotézy, kterou L vytvoří na základě množiny atributů $A \cup \{a_i\}$ je větší než přesnost dosažená pomocí množiny atributů A.
Tento pojem je přirozený zvláště pro algoritmy pro výběr atributů, které prohledávají prostor podmnožin atributů tak, že inkrementálně zvětšují (či zmenšují) aktuální množinu atributů.

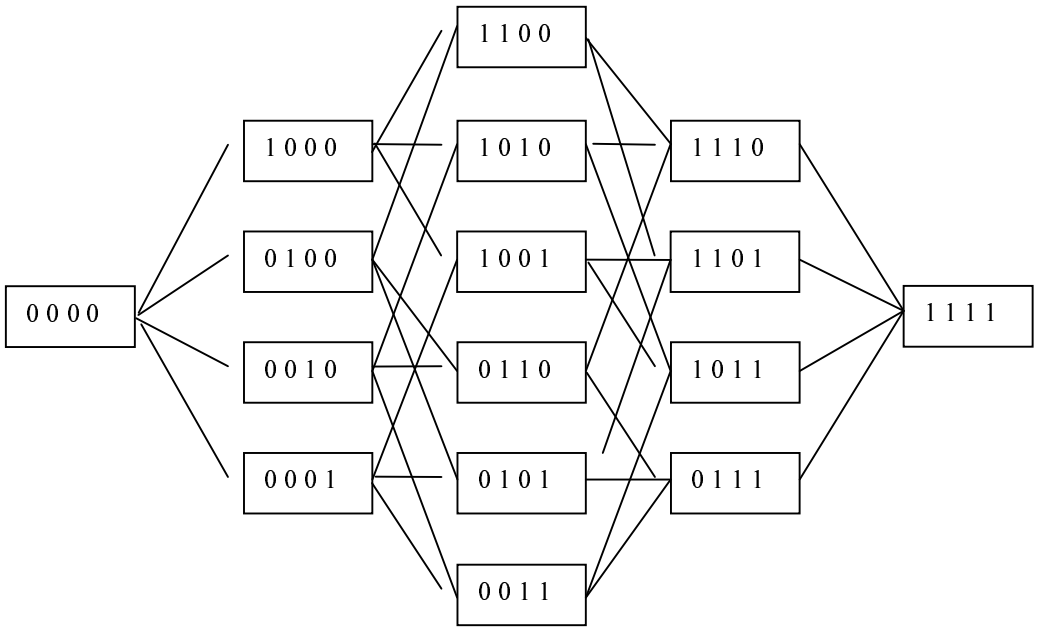
K ilustraci uvedených definic relevance nám poslouží následující jednoduchý příklad.

Příklad. Uvažujme pojmy, které se dají vyjádřit jako disjunkce atributů (například $a_1 \vee a_3 \vee a_7$) a předpokládejme, že algoritmus učení má k dispozici následující tréninkové příklady :

x_1	x_{11}	x_{21}	x_{30}	
1 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0		+
1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0		+
0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0		+
0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1		+
0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0		–

Objekty jsou tedy popsány třiceti (dvouhodnotovými) atributy $a_1, \ldots, a_{30}$ s hodnotami $x_1, \ldots, x_{30}$; cílová funkce je také dvouhodnotová, instancím přiřazuje hodnoty + a – . Atributy relevantní podle definice 1 budou záviset na cílové funkci (je však vidět, že každá konzistentní disjunkce musí obsahovat a_1). Co se týče definice 2 a definice 3, a_1 je silně relevantní a ostatní atributy jsou slabě relevantní (například a_2 je slabě relevantní, protože se stane silně relevantní po odstranění $a_1, a_3, \ldots, a_{10}$). V souvislosti s definicí 4 je $r(S,C) = 3$, protože 3 je počet atributů relevantních nejkratší konzistentní disjunkci. Pojem inkrementální užitečnosti z definice 5 závisí na konkrétním algoritmu; je ale vidět, že při dané množině atributů $A = \{a_1, a_2\}$ není a_3 užitečný, ale některý (kterýkoliv) z $a_{11}, \ldots, a_{30}$ je užitečný.

V dalším se budeme zabývat algoritmy pro výběr atributů, obecněji algoritmy pro zpracování dat obsahujících velký počet irelevantních atributů. Odpovídajícím rámcem pro popis takových postupů (zvláště jde-li o explicitní výběr atributů) je heuristické prohledávání; každý stav v prohledávaném prostoru určuje nějakou podmnožinu (možných) atributů. Pro ilustraci si můžeme představit následující jednoduchý stavový prostor.



Obr. 1

K dispozici tedy máme čtyři atributy a_1, a_2, a_3, a_4; stavem je čtveřice nul a jedniček, kde jednička resp. nula na i-tém místě ($i = 1,\ldots,4$) znamená, že příslušná podmnožina atributů obsahuje resp. neobsahuje atribut a_i. Jinými slovy, $\langle 0, 0, 0, 0 \rangle$ značí prázdnou množinu atributů, $\langle 1, 0, 1, 0 \rangle$ je kód pro množinu $\{a_1, a_3\}$ a $\langle 1, 1, 1, 1 \rangle$ odpovídá množině $\{a_1, a_2, a_3, a_4\}$ všech atributů.

Charakter prohledávacího prostoru je určen :

- volbou počátečního stavu. Ta má vliv na směr prohledávání a operátory použité pro generování následných stavů. Jak ukazuje obrázek 1,existuje přirozené (částečné) uspořádání na tomto prostoru, kde každý synovský uzel má o jeden atribut více, než jeho rodičovské uzly. Můžeme tedy začít s prázdnou množinou a postupně přidávat atributy (dopředný výběr); podobně bychom mohli začít se všemi atributy a postupně je ubírat (zpětná eliminace);
- strategií prohledávání. Exhaustivní prohledávání nepřichází v úvahu (stavový prostor obsahuje 2^n možných podmnožin n atributů). Často se užívá nějaké formy hladového algoritmu (v každém stavu uvážíme možné změny aktuální množiny atributů, jednu vybereme a uskutečníme), například horolezecké metody postupné selekce či eliminace (můžeme generovat všechny následníky a vybrat nejlepšího nebo vybrat prvního, který zlepšuje přesnost na dané tréninkové množině). Dá se též užít některá sofistikovanější metoda, například best-first prohledávaní;
- volbou ohodnocovací funkce užité pro ohodnocování alternativních podmnožin atributů. Obvykle používaná míra zahrnuje schopnost atributů rozlišovat mezi třídami (tj. hodnotami cílové funkce), které se objevují v tréninkových datech. Řada algoritmů užívá kriterií založených na teorii informace, jiné přímo měří přesnost dosahovanou na tréninkové množině nebo zvláštní testovací množině;
- podmínkou ukončení. Přidávání nebo ubírání atributů můžeme ukončit, jestliže žádná takto vzniklá alternativa aktuálního stavu nezlepší odhad přesnosti klasifikace. Jinou možností je pokračovat tak dlouho, dokud se přesnost nebude zhoršovat, případně můžeme pokračovat, dokud neprojdeme celý prostor „až na druhou stranu“ a pak vybrat nejlepší stav (množinu atributů).

Vraťme se nyní k situaci, kdy se učíme pojem vyjádřitelný pomocí disjunkce dvouhodnotových atributů – viz výše uvedený příklad. Cílová funkce rozdělila tréninkové příklady na pozitivní (ve kterých má hodnotu +) a negativní (s hodnotou −). Popíšeme jednoduchý tzv. hladový pokrývací algoritmus, který vytváří disjunkci (hypotézu), aproximující cílovou funkci. Hledaná disjunkce bude také klasifikovat instance z tréninkových příkladů; jde o to, aby to dělala „správně“, tedy stejně, jako cílová funkce (jinými slovy, aby byla konzistentní s tréninkovou množinou). Uvědomme si nejprve, že pokud pro atribut a_i platí, že v žádné instanci patřící negativnímu příkladu není na i-tém místě hodnota 1 (názorně budeme říkat, že a_i se nevyskytuje v žádném negativním příkladu), je a_i „bezpečný“ z hlediska vytváření hypotézy – žádná disjunkce, která ho obsahuje, nemůže „jeho vinou“ chybně ohodnotit žádný negativní příklad.

Hladový pokrývací algoritmus pracuje v následujících krocích :

- vezmeme prázdnou disjunkci (definitoricky dává hodnotu „negativní“ pro každý příklad)
- z atributů, které se nevyskytují v žádném negativním příkladu (a jsou tedy bezpečné z hlediska přidání do hypotézy) vybereme takový, jehož přidáním se nejvíce zvýší počet správně klasifikovaných pozitivních příkladů
- opakujeme, pokud existuje nějaký bezpečný atribut zvyšující počet správně klasifikovaných pozitivních příkladů

Podíváme-li se na obrázek 1, vidíme, že algoritmus začne pracovat ve stavu $\boxed{0, 0, 0, 0}$, pohybuje se pouze doprava, ohodnocuje podmnožiny atributů (stavy) na základě chování příslušných disjunkcí na tréninkové množině (přičemž dává „nekonečně velkou pokutu“ za chybně klasifikované negativní příklady) a končí, pokud žádný další krok (žádné přidání atributu) nemůže ostře zlepšit chování aktuálního stavu, tj. podmnožiny atributů. Pro data z našeho příkladu algoritmus vybere jako první atribut a_1, pak (asi) a_{11}, pak (asi) a_{21} a pak skončí. Snadno se nahlédne, že pokud existuje disjunkce atributů konzistentní s tréninkovou množinou, uvedený algoritmus ji najde. Dále se dá ukázat, že počet atributů vybraných tímto způsobem je nejvýše $O(log|S|)$-krát větší než počet relevantních atributů z definice 4 (to plyne z faktu, že v průběhu činnosti algoritmu musí vždy existovat atribut, jehož přidáním ubude alespoň $1/r(S, C)$-tina z počtu chybně klasifikovaných pozitivních příkladů). Na druhé straně, nalezení nejkratší disjunkce konzistentní s tréninkovou množinou je NP-těžká úloha. Podrobněji se o těchto věcech lze dozvědět v [12] , [14] a [11].

Algoritmus můžeme použít též ke srovnání různých definic relevance :

- inkrementálně užitečné atributy (definice 5) jsou také slabě relevantní (definice 3); opak nemusí vždy platit. Ve skutečnosti
- nejsou-li data konzistentní s žádnou disjunkcí, pak i silně relevantní atributy (definice 2) může algoritmus ignorovat (díky svému konzervativnímu charakteru – ignoruje každý atribut, který by mohl způsobit chybnou klasifikaci negativního příkladu);
- naopak jsou-li data konzistentní s nějakou disjunkcí, pak všechny silně relevantní atributy jsou inkrementálně užitečné (a budou tedy algoritmem vybrány), i když algoritmus může preferovat slabě relevantní atributy před silně relevantními (to závisí na jeho ohodnocovacím kriteriu).

V dalším textu se budeme zabývat třemi základními přístupy k výběru atributů.

1.2. Vnořování (embedded approaches).

Metody vytváření logických popisů jsou příkladem metod, které vnořují výběr atributů do základního induktivního učícího algoritmu. Řada algoritmů vytvářejících logické konjunkce, viz například [26], [34] nebo [35] (a také uvedený hladový pokrývací algoritmus) dělá o něco více, než že k či z popisu pojmu přidává či ubírá atributy v závislosti na předpovědi chyby na nových instancích. Využívají (částečného) uspořádání stavů k řízení prohledávání (jehož cílem je najít popis pojmu). Teoretické výsledky pro učení čistě konjunktivních nebo čistě disjunktivních pojmů jsou povzbuzující (viz hladový pokrývací algoritmus). Zdá se navíc, že můžeme dostat ještě o něco menší meze v PAC-učení (tedy ještě o něco lepšího odhadu pro požadovanou minimální velikost vzorku), pokud algoritmus skončí „o něco dříve" (a některé tréninkové příklady zůstanou chybně klasifikovány). Vzhledem k tomu, že výsledné hypotézy jsou pak „docela malé", velikost vzorku roste pouze logaritmicky s počtem irelevantních atributů. Tyto výsledky se dají aplikovat i na situace, kdy cílový pojem je charakterizován jako konjunkce (nebo disjunkce) seznamu funkcí, vytvořených induktivním algoritmem.

Podobné operace přidávání a odebírání atributů tvoří jádro metod pro učení složitějších logických pojmů, používajících dále rutiny pro kombinování atributů do složitějších popisů. Například rekurzivní metody vytvářející rozklad (jako jsou ID3 nebo C4.5 – viz [28] a [29]) provádějí hladové prohledávání prostoru rozhodovacích stromů a v každém kroku užívají ohodnocovací funkce k výběru atributu s nejlepší schopností rozlišovat mezi třídami. Rozklad tréninkových dat je založen na vybraném atributu a celý proces se opakuje pro podmnožiny; tím se zvětšuje rozhodovací strom. Proces končí, není-li už další rozlišení možné. Tyto techniky rekurzivního hladového pokrývání byly použity i pro vytváření složitějších formulí, například tzv. *k*-členných DNF formulí ([9]).

Podobným způsobem je selekce atributů vnořena i do „rozděl-a-panuj" metody učení rozhodovacích seznamů (viz například [7] , [24] , [27] , [30]). Ta vybírá atributy, přispívající k rozlišení třídy c (jejíž popis se snažíme naučit) od ostatních. Vybrané atributy přidáváme do konjunkce (obvykle se zajímáme pouze o konjunkce, jejichž délka je předem omezena číslem k), s cílem získat konjunkci (test) takovou, že všechny tréninkové příklady, pro které je pravdivá, mají „stejné znaménko" (jsou tedy všechny pozitivní nebo všechny negativní). Označíme podmnožinu těchto příkladů T, do rozhodovacího seznamu pro c přidáme test spolu s příslušnou klasifikací příkladů z T a celý proces opakujeme pro množinu $S - T$ zbývajících tréninkových příkladů.

Obě tyto metody (vytváření rozkladu a rozděl-a-panuj) explicitně vybírají atributy pro zařazení do větve stromu či do pravidla (konjunkce) na základě jejich preference před jinými (méně relevantními nebo irelevantními) atributy. Z tohoto důvodu se dá očekávat, že budou dobře pracovat v oblastech, kde se vyskytuje mnoho irelevantních atributů. Experimentální studie naznačují, že metody, užívající rozhodovacích stromů, závisí pro některé druhy cílových pojmů (například pro logické konjunkce) na počtu irelevantních atributů lineárně; pro jiné pojmy však vykazují exponenciální růst stejně jako učení metodou nejbližšího souseda. Standardní vysvětlení tohoto efektu uvádí jako důvod závislost těchto algoritmů na hladovém výběru atributů, které vzájemně rozlišují různé třídy. Tento přístup funguje dobře, pokud jsou mezi relevantními atributy pouze malé interakce (jako je tomu u konjunktivních pojmů); naopak přítomnost vzájemných interakcí mezi atributy (která může mít za následek, že relevantní atribut sám o sobě se nejeví lépe rozlišující než irelevantní) může při tomto schématu hladového výběru působit značné problémy.

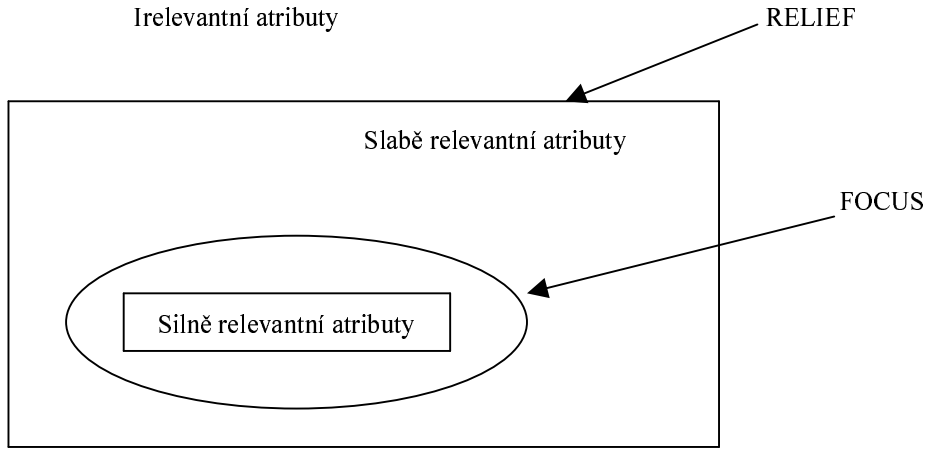
1.3. Filtrace (filter approaches).

Jedná se o samostatný proces výběru atributů předcházející vlastní činnosti induktivního učícího algoritmu (před vlastním učením se odfiltrují irelevantní atributy). Je založen na vlastnostech treningové množiny a nezávisí na induktivním algoritmu učení.

Nejjednodušší filtrovací metodou je ohodnocování jednotlivých atributů v závislosti na jejich korelaci s cílovou funkcí a výběru k nejlepších (s nejvyšší hodnotou). Běžně se užívá při úkolu charakterizace textu, často v kombinaci s bayesovským učením nebo s učením metodou nejbližšího souseda.

Algoritmus RELIEF (viz [16] , [17]) uplatňuje při výběru atributů stejné základní schéma s tím, že užívá složitější ohodnocovací funkce. Atributům přiřazuje váhy, značící míru jejich relevance vůči cílovému pojmu. Jinými slovy ohodnocuje (váží) atributy podle toho, jak dobře jejich hodnoty dokáží rozlišovat „blízké" instance; RELIEF je inspirován učením metodou nejbližšího souseda. Je to pravděpodobnostní algoritmus, který náhodně vybírá příklady z tréninkové množiny a průběžně aktualizuje hodnoty relevance v závislosti na rozdílech mezi vybraným příkladem a dvěma jemu nejbližšími příklady (jednoho se stejnou, jednoho s opačnou klasifikací). Snaží se najít všechny relevantní atributy, i když řada (slabě relevantních) z nich může být redundantní a k popisu cílového pojmu by ve skutečnosti stačila pouze část z nich. Po ukončení filtrace používá RELIEF algoritmus ID3 k vytvoření rozhodovacího stromu, přičemž pro popis tréninkových příkladů i pro chod ID3 se užívá pouze vybraných atributů. Původní verze algoritmu RELIEF pracovala s diskrétními i spojitými atributy avšak pouze s dvouhodnotovou cílovou funkcí. V [20] je popsáno zobecnění na případ neúplných dat (eventuelně s šumem) a vícehodnotové cílové funkce.

Algoritmus FOCUS ([1] , [2]) hledá minimální množinu atributů, která stačí pro určení správné klasifikace všech příkladů z tréninkové množiny. Postupně vyšetřuje všechny jedno - , dvou - atd. prvkové množiny atributů až narazí na podmnožinu takovou, že žádné dva příklady se stejnou kombinací hodnot těchto atributů nemají různou klasifikaci. Pak opět zavolá algoritmus ID3 (na celou tréninkovou množinu) s použitím pouze vybraných atributů. Při porovnávání s přímým sestrojováním rozhodovacího stromu (při daném počtu tréninkových příkladů na náhodně vybraných cílových pojmech) se ukazuje, že FOCUS téměř nereaguje na přidání irelevantních atributů, zatímco přesnost rozhodovacích stromů se výrazně snižuje. Obrázek 2 názorně porovnává zacílení a dosah obou zmíněných algoritmů.

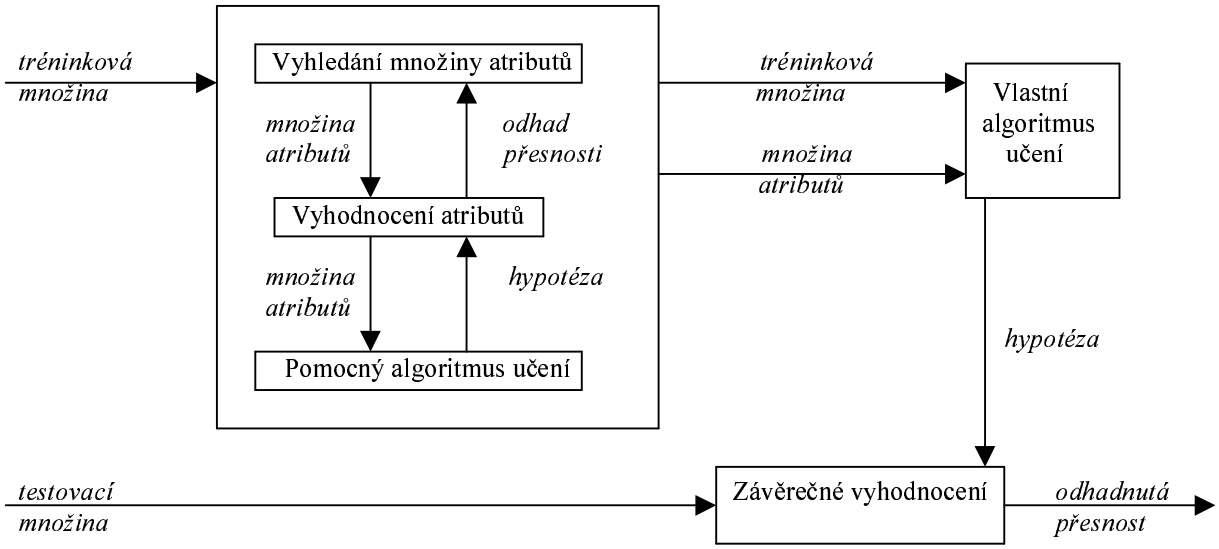


Obr.2

Závěrem ještě dvě poznámky. Řekli jsme, že filtrační přístup k výběru atributů je nezávislý na (poté použitém) induktivním učícím algoritmu. To však může někdy přinášet určité problémy, pokud ho aplikujeme slepě, tedy bez vztahu k učenému pojmu. Jestliže například FOCUS preferuje minimální množiny atributů, pak v situaci, kdy mezi atributy popisujícími pacienty je také jejich rodné číslo, FOCUS tento atribut najde a skončí (rodné číslo samo jednoznačně určuje pacienta); přitom ovšem rodné číslo nikterak neumožňuje určit pacientovu diagnózu (kterou jsme se chtěli naučit), což má za následek, že jakýkoli induktivní algoritmus učení bude pracovat velmi špatně. Nakonec poznamenejme, že pro filtrování se dají použít i rozhodovací stromy; v rozhodovacím stromu se typicky vyskytují pouze některé atributy a ty mohou sloužit jako vstupní množina atributů pro učení metodou nejbližšího souseda (viz [6]). I při takovémto postupu je třeba jisté opatrnosti (nefunguje stejně dobře pro libovolná data). Atributy „dobré“ pro vytváření rozhodovacího stromu nemusí být nutně užitečné pro učení metodou nejbližšího souseda. Navíc zatímco učení metodou nejbližšího souseda může s užitkem brát v úvahu efekt mnoha relevantních atributů, při úvodním vytváření rozhodovacího stromu se často použije (a k vlastnímu učení poskytne) pouze malý počet atributů.

1.4. Obalování (wrapper approaches).

Jde o proces výběru atributů (viz [13]) probíhající mimo vlastní induktivní algoritmus učení, ale používající nějakou (obecně jinou) induktivní učící metodu jako subrutinu. Tato induktivní metoda se používá jako black box ; nemáme o ní žádné znalosti, zajímá nás pouze výstup. Algoritmus výběru množiny atributů prohlíží týž prostor jako vnořovací a filtrační metody a hledá „dobrou“ podmnožinu následujícím způsobem. Na vyčleněných tréninkových a testovacích datech nechá pracovat (pomocný) induktivní učící algoritmus opakovaně pro různé množiny atributů. Odhad přesnosti naučené funkce (hypotézy, pravidla přiřazujícího klasifikaci) pak slouží jako míra aktuálně vyšetřované množiny atributů (vnitřní induktivní algoritmus je vlastně součástí ohodnocovací funkce). Podmnožina atributů s nejlepším ohodnocením je (spolu s celou tréninkovou množinou) vstupem pro vlastní induktivní algoritmus učení. Jím vytvořená výsledná funkce je pak ohodnocena pomocí nezávislé testovací množiny, která nebyla použita v procesu výběru atributů. Celý obalovací přístup si můžeme schematicky znázornit následujícím způsobem.



Obr. 3

Obecný argument ve prospěch takovéhoto přístupu je, že induktivní metoda, která bude používat podmnožinu atributů, může vést k lepšímu odhadu přesnosti klasifikátoru (tedy k lepšímu odhadu „dobroty“ podmnožiny atributů) než nějaká samostatná míra (která může mít zcela jiné apriorní omezení). Jeho hlavní nevýhodou (ve srovnání s filtračními metodami) je výpočetní náročnost plynoucí z toho, že se pomocný induktivní algoritmus volá znovu pro každou vyšetřovanou množinu atributů (to vede k vytváření technik urychlujících proces ohodnocování). Dá se očekávat, že postupy jako je učení metodou nejbližšího souseda (které – není-li řečeno jinak – berou do úvahy všechny atributy) budou mít z obalovacího výběru atributů větší užitek než algoritmy, které samy obsahují vnořená schémata; toto očekávání vedlo k tomu, že podstatná část práce byla věnována obalovacím postupům pro učení metodou nejbližšího souseda.

Příkladem těchto snah je algoritmus OBLIVION (viz [22]), který kombinuje obalovací ideu s učením jednoduchou metodou nejbližšího souseda. Proces výběru atributů efektivně mění metriku, užívanou při učení metodou nejbližšího souseda pro stanovení klasifikace nové instance tím, že bere do úvahy pouze relevantní atributy a ignoruje ostatní. OBLIVION provádí zpětné prohledávání podmnožin atributů (zpětnou eliminaci); začíná s množinou všech atributů a iterovaně odstraňuje ten, jehož odebrání vede k největšímu zlepšení odhadu přesnosti (pokud by odebral relevantní atribut, přesnost by naopak klesla). Takto pokračuje do té doby, kdy po odebrání libovolného atributu by přesnost vždy klesla (tedy pokud mezi atributy je alespoň jeden irelevantní). Do našeho schématu znázorňujícího obalovací přístup se dá OBLIVION zahrnout zřejmě tak, že za vlastní (vnější) i pomocný (vnitřní) induktivní algoritmus dosadíme algoritmus učení metodou nejbližšího souseda.

1.5. Vážení atributů.

Dosud jsme se zabývali algoritmy, které explicitně vybíraly „nejvíce relevantní“ podmnožinu atributů. Je možný i jiný přístup (zvláště pro vnořovací metody), při němž se atributy opatřují váhami, vyjadřujícími stupeň jejich relevance. Patrně nejznámější takovou metodou užívající vážení atributů je učení pomocí lineárního klasifikátoru, známé jako Perceptron algoritmus (viz [31] nebo [25]). Lineární klasifikátor je reprezentován dvojicí $(\boldsymbol{w}, \Theta)$, kde $\boldsymbol{w}$ je n-tice reálných čísel $w_1, \ldots, w_n$ (váhový vektor) a Θ reálné číslo (mez nebo práh). Máme-li n binárních atributů $a_1, \ldots, a_n$, pak pro instanci $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle \in \{0,1\}^n$ vydá klasifikátor hodnotu 1, pokud $\mathbf{w}.\mathbf{x} \geq \Theta$ a hodnotu 0 jindy. Poznamenejme, že speciálním případem lineárního klasifikátoru je disjunkce $a_{i_1} \vee \ldots \vee a_{i_r}$ délky r. Můžeme ji reprezentovat jako dvojici $(\mathbf{w}, ½)$, kde $w_{i_1} = \ldots = w_{i_r} = 1$ a $w_j = 0$ pro $j \notin \{i_1, \ldots, i_r\}$. Perceptron je aditivní algoritmus (mění váhy přičítáním nebo odečítáním konstanty), což může vést k potížím v zadáních s převahou irelevantních atributů.

V této souvislosti vznikl alternativní algoritmus WINNOW (viz [23]), který je multiplikativní (váhy se mění násobením nebo dělením konstantou větší než jedna). WINNOW se týká učení disjunkcí délky r (souvisí s naší definicí 4 relevance jakožto míry složitosti – “správná” disjunkce, která správně říká, které příklady jsou pozitivní a které negativní, má délku r, tj. je tvaru $a_{i_1} \vee \ldots \vee a_{i_r}$). Má-li algoritmus nějak nastavené váhy a pomocí nich (a prahu) ohodnotí danou instanci odlišně v porovnání se správnou disjunkcí, říkáme, že udělal chybu.

WINNOW (základní verze).

1) Inicializujeme váhy $w_1, \ldots, w_n$ atributů $a_1, \ldots, a_n$ hodnotou 1
2) Pro danou instanci $\langle x_1, \ldots, x_n \rangle$ vydáme výstupní hodnotu 1, pokud $w_1.x_1 + \ldots + w_n.x_n \geq n$ a výstupní hodnotu 0 jindy
3) Pokud algoritmus udělal chybu, pak
   a) jestliže vydal 0 v pozitivním příkladu, pak pro každé i, pro které $x_i = 1$, zdvojnásobíme w_i
   b) jestliže vydal 1 v negativním příkladu, pak pro každé i, pro které $x_i = 1$, vydělíme w_i dvěma
4) Jdeme do 2.

WINNOW postupně probírá instance příslušející příkladům z tréninkové množiny S (jde tedy o cyklus typu „while $S \neq 0$ do ... "). Po jejím vyčerpání by měly být váhy nastaveny (naučeny) tak, aby lineární klasifikátor, který tyto váhy (spolu s prahovou hodnotou n) definují, „dobře aproximoval" cílovou disjunkci délky r. Pro takovýto algoritmus je jistě důležité umět odhadnout počet chyb, které udělá během procesu učení při jistých vstupních posloupnostech tréninkových příkladů (se zřejmým záměrem navrhnout algoritmus tak, aby se tento počet chyb minimalizoval). Řekneme, že příklad je konzistentní s disjunkcí $a_{i_1} \vee \ldots \ldots \vee a_{i_r}$ délky r, pokud :

- je-li pozitivní, má alespoň na jednom z míst $i_1, \ldots, i_r$ hodnotu 1
- je-li negativní, má na všech těchto místech hodnotu 0

Věta. Pro každou posloupnost příkladů konzistentních s disjunkcí délky r udělá WINNOW nejvýše $2 + 3r(1 + \log n)$ chyb.

Důkaz tohoto tvrzení pro zajímavost stručně naznačíme (viz [3]). Nejprve odhadneme počet chyb, které algoritmus udělá při ohodnocování pozitivních příkladů. Každá taková chyba zdvojnásobí alespoň jednu váhu w_j pro $j \in \{i_1, \ldots, i_r\}$ (tzv. relevantní váhu); současně chyba při ohodnocení negativního příkladu nebude půlit žádnou relevantní váhu. Dále uvážíme, že každá z relevantních vah může být zdvojnásobena nejvýše $(1+\log n)$-krát (mohou se zdvojnásobovat pouze váhy menší než n). WINNOW tedy udělá na pozitivních příkladech nejvýše $r(1 + \log n)$ chyb. Při odhadu počtu chyb na negativních příkladech budeme pracovat s celkovou vahou (tedy součtem všech w_i pro $i = 1, \ldots, n$); ta byla inicializována hodnotou n. Každá chyba na pozitivním příkladu zvětší celkovou váhu nejvýše o n (před zdvojnásobováním muselo být $w_1.x_1 + \ldots + w_n.x_n < n$); každá chyba na negativním příkladu zmenší celkovou váhu alespoň o $n/2$ (před půlením muselo být $w_1.x_1 + \ldots + w_n.x_n \geq n$); celková váha přitom nemůže být nikdy záporná. Odtud plyne, že počet chyb na negativních příkladech je nejvýše dvojnásobkem počtu chyb na pozitivních příkladech plus 2, tedy nejvýše $2 + 2r(1 + \log n)$. Sečtením odhadů pro počet chyb, které algoritmus udělá při ohodnocování pozitivních a negativních příkladů, dostaneme proklamovaný výsledek.

Je tedy vidět, že je-li délka r cílové konjunkce (neboli počet relevantních atributů) konstantní, chování WINNOW (počet chyb) se zhoršuje pouze logaritmicky s počtem n všech atributů (tedy také s počtem irelevantních atributů). Naproti tomu pro aditivní algoritmy (jako je Perceptron) platí, že pro každé $r \leq n$ udělají v uvažované situaci (učení se disjunkci délky nejvýše r) alespoň $(n + r - 1)/2$ chyb. Jejich chování se tedy s počtem irelevantních atributů zhoršuje alespoň lineárně. Podrobněji se vlastnosti těchto algoritmů studují v [18].

2. Výběr relevantních příkladů.

Tak jako jsou některé atributy užitečnější než jiné, stejně i některé příklady mohou lépe řídit proces učení než jiné. Odtud plyne zájem o metody výběru příkladů. Necháme stranou „laskavého učitele", který dává informativní příklady či ideální posloupnosti tréninkových příkladů (jako je P. H. Winston a jeho „near misses" v [35]) a budeme si všímat robustnějších metod, při nichž učící se systém sám vybírá příklady. Můžeme uvést alespoň tři situace, v nichž existují dobré důvody pro vybírání (či eliminaci) příkladů užitých během procesu učení :

- je-li učící algoritmus výpočetně náročný; pak je rozumné (z důvodu výpočetní efektivity) učit se pouze z některých příkladů;
- je.li snadno k dispozici mnoho neklasifikovaných instancí, ale cena získání jejich klasifikace je vysoká (například proto, že klasifikaci dodává expert);
- chceme-li zvýšit účinnost učení tím, že zaměříme pozornost pouze na „informativní" příklady (čím řídíme prohledávání v prostoru hypotéz).

Rozlišujeme zde příklady relevantní z hlediska informace, kterou nesou či algoritmu, který je používá (většinou se preferuje druhé hledisko). Metody výběru můžeme opět dělit na vnořené, filtrační a obalovací; hlavním kriteriem ale pro nás bude, zda vybíráme příklady z ohodnocených (klasifikovaných) nebo neohodnocených instancí.

2.1. Výběr z ohodnocených příkladů.

Ne všechny příklady jsou pro učení stejně užitečné. Proces jejich výběru může být součástí různých algoritmů.Některé z nich (příkladem je WINNOW) se učí z tréninkového příkladu pouze tehdy, pokud ho aktuální hypotéza klasifikovala chybně. Takovým vnořeným metodám se říká konzervativní (ignorují všechny příklady, které jejich hypotézy klasifikují správně, tedy shodně s cílovou funkcí). Předpokládáme-li, že tréninková i testovací data jsou vybrána pomocí téhož pevného rozložení pravděpodobnosti (při stejné distribuční funkci na množině X všech instancí) – tento předpoklad se někdy nazývá předpokladem stacionárnosti - máme s vysokou pravděpodobností zaručeno, že tréninková data dobře odpovídají kriteriím užitým pro testování. S pokračujícím učením však znalost o některých částech vstupního prostoru instancí vzrůstá a příklady z této „lépe poznané“ části začínají být méně užitečné. Například má-li konzervativní algoritmus chybu 20 %, ignoruje 80 % tréninkových příkladů; dosáhne-li chyby 10 %, ignoruje jich 90 %.

V PAC-modelu učení (tohoto pojmu jsme se dotkli již v úvodní kapitole našeho textu) se řeší otázka, zda lze stanovit minimální počet tréninkových příkladů tak, aby učící algoritmus na jejich základě vydal hypotézu, jejíž chyba je s vysokou pravděpodobností dostatečně malá. Uvažujme situaci, kdy učící algoritmus se učí pojmu $c : X \rightarrow \{0, 1\}$; k dispozici má tréninkovou množinu S příkladů $\langle x, c(\boldsymbol{x})\rangle$, kde příslušné instance $\boldsymbol{x}$ jsou vybrány náhodně z množiny instancí X v souladu s distribuční funkcí D (definované na X). Výsledkem činnosti algoritmu je hypotéza $h : X \rightarrow \{0, 1\}$, konzistentní s S. Pro danou instanci $\boldsymbol{x}$ můžeme hodnotu $h(\boldsymbol{x})$ chápat jako randomizovanou předpověď klasifikace $\boldsymbol{x}$, která se rovná 1 s pravděpodobností $h(\boldsymbol{x})$ a rovná se 0 s pravděpodobností $1 – h(\boldsymbol{x})$. Chybou ε hypotézy h pak rozumíme pravděpodobnost chybné předpovědi, tedy $\varepsilon = \Pr (h(\boldsymbol{x}) \neq c(\boldsymbol{x}) \mid \boldsymbol{x}$ vybráno náhodně z X v souladu s $D)$. Dá se spočítat (viz například [4]), že pro daná (malá) čísla $\varepsilon , \delta > 0$ platí pro počet m tréninkových příkladů nerovnost $m \geq 1/\varepsilon (\log |H| + \log 1/\delta)$, kde $|H|$ je velikost množiny možných hypotéz. Vydá-li tedy učící algoritmus hypotézu konzistentní s (alespoň) tímto počtem tréninkových příkladů, pak tato hypotéza má s pravděpodobností alespoň $1 - \delta$ chybu nejvýše ε. Počet požadovaných příkladů (minimální velikost tréninkové množiny) je funkcí ε a δ a nazývá se složitost vzorku.

V PAC-modelu tedy učící algoritmus potřebuje zhruba zdvojnásobit počet prozkoumaných příkladů, chce-li dosáhnout poloviční chyby. Pro konzervativní algoritmy, protože počet příkladů skutečně použitý pro učení je úměrný chybě, zůstává počet nových příkladů použitých algoritmem v situaci, kdy chce zmenšit chybu na polovinu, zhruba stejný. To znamená, že počet příkladů skutečně použitých k dosažení chyby ε roste v $1/\varepsilon$ reálně spíše logaritmicky než lineárně. I když tyto výsledky platí pouze pro konzervativní algoritmy, které vnořují výběr příkladů do procesu učení, můžeme užít explicitního výběru příkladů k dosažení podobného efektu i u jiných induktivních metod.

Příkladem je obalovací metoda zvaná „boosting“. Je to obecná metoda, která má své kořeny také v modelu PAC-učení a která se snaží zlepšit přesnost daného (generického) algoritmu. Silný PAC-učící algoritmus je takový algoritmus řízeného induktivního učení z příkladů, který pro dané $\varepsilon , \delta > 0$ a tréninkovou množinu S vydá s pravděpodobností alespoň $1 - \delta$ konzistentní hypotézu h, mající chybu nejvýše ε ; přitom čas spotřebovaný algoritmem musí být polynomiální v $1/ \varepsilon , 1/ \delta$ a dalších relevantních parametrech (jako je „velikost“ tréninkových příkladů, „velikost“ nebo „složitost“ cílového pojmu). Slabý PAC-algoritmus se od silného liší pouze tím, že $\varepsilon \leq ½ - \gamma$, kde $\gamma \geq 0$. Jinými slovy, slabý algoritmus pracuje jen „o něco lépe“, než kdyby výsledná hypotéza určovala klasifikaci náhodně (protože cílová funkce je dvouhodnotová, měla by hypotéza h_n, která by instancím přiřazovala jejich klasifikaci náhodně, chybu ½; člen γ říká, o kolik je předpověď klasifikace vydaná výslednou hypotézou h lepší než náhodná). Cílem metody „boosting“ je transformovat daný (libovolně zvolený) slabý PAC-algoritmus na silný (dokonce takový, že jím vydaná výsledná hypotéza má libovolně malou chybu).

V [33] je popsán algoritmus AdaBoost (adaptivní boosting) a jsou tam stručně shrnuty jeho vlastnosti. (AdaBoost byl prvně uveden v [10], kde se celá problematika vyšetřuje v širším rámci a podrobněji.) Má (alespoň ve své základní podobě) jako vstup tréninkovou množinu S o m prvcích $\langle \boldsymbol{x}_1, c(\boldsymbol{x}_1)\rangle$, ..., $\langle \boldsymbol{x}_m, c(\boldsymbol{x}_m)\rangle$, kde každé $\boldsymbol{x}_i$ je instance z prostoru X všech instancí a hodnota (klasifikace) $c(\boldsymbol{x}_i)$ je 1 nebo -1; c je binární funkce (pojem), kterou se učíme. Množina S byla vybrána náhodně v souladu s nějakou pevnou (nám neznámou) distribuční funkcí na X (přesněji, označíme-li $Y = \{-1, 1\}$, jde o distribuční funkci na $X \times Y$). Volá daný slabý (generický) algoritmus opakovaně v krocích $t = 1$, ... ,T. V každém kroku t pracuje slabý algoritmus s distribuční funkcí D_t na S (kterou si v průběhu činnosti sám postupně upravuje) a vydá slabou hypotézu $h_t : X \rightarrow \{-1, 1\}$. (To, že za možné hodnoty cílové funkce a hypotézy bereme nyní -1 a 1 místo 0 a 1, není samozřejmě podstatné; pouze nám to umožní převzít jednodušší popis algoritmu z [33].)

AdaBoost

Vstup : $\langle \boldsymbol{x}_1, c(\boldsymbol{x}_1)\rangle$, ... ,$\langle \boldsymbol{x}_m, c(\boldsymbol{x}_m)\rangle$, kde $\boldsymbol{x}_i \in X$, $c(\boldsymbol{x}_i) \in \{-1, 1\}$ pro všechna $i = 1, \ldots, m$

Inicializace : $D_1 = 1/m$

Pro $t = 1, \ldots, T$:

- nech běžet slabý algoritmus užívající distribuční funkci D_t
- získej od něj slabou hypotézu h_t, která má chybu $\varepsilon_t = \Pr_{i\sim D_t}(h(\boldsymbol{x}_i) \neq c(\boldsymbol{x}_i))$
- polož $$\alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$
- aktualizuj

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{jestliže } h_t(\boldsymbol{x}_i) = c(\boldsymbol{x}_i) \\ e^{\alpha_t} & \text{jestliže } h_t(\boldsymbol{x}_i) \neq c(\boldsymbol{x}_i) \end{cases}$$
$$= \frac{D_z(i) \times \exp(-\alpha_t c(\boldsymbol{x}_i) h_t(\boldsymbol{x}_i))}{Z_t}$$

kde Z_t je normalizační faktor (zvolený tak, aby D_{t+1} byla distribuční funkce na S)

Výstup : finální hypotéza $\mathrm{H}(\boldsymbol{x}) = \mathrm{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$

Algoritmus tedy pracuje s distribuční funkcí (nebo chceme-li s váhami) na tréninkové množině S (hodnota funkce D_t v i-tém příkladu $\langle \boldsymbol{x}_i, c(\boldsymbol{x}_i)\rangle$ tj. váha, kterou D_t přiřadí tomuto příkladu, se značí $D_t(i)$); na počátku jsou váhy všech příkladů stejné. Slabá hypotéza h_t, kterou vydá v kroku t, má chybu $\varepsilon_t = \Pr_{i\sim D_t}(h(\boldsymbol{x}_i) \neq c(\boldsymbol{x}_i)) = \sum_{i:h_t(\boldsymbol{x}_i)\neq c(\boldsymbol{x}_i)} D_t(i)$; tato chyba je samozřejmě závislá na funkci D_t, kterou používá v kroku t slabý algoritmus. Parametr α_t měří důležitost připisovanou hypotéze h_t ; je vidět, že pro $\varepsilon_t < ½$ je $\alpha_t \geq 0$ a že α_t roste s klesajícím ε_t. Funkce D_t se následně aktualizuje tak, že vzrůstají váhy příkladů chybně klasifikovaných hypotézou h_t a naopak klesají váhy těch příkladů, které h_t klasifikovala správně. To znamená, že úpravou vah se pozornost algoritmu soustřeďuje především na „aktuálně těžké“ příklady (přesnost aktuálně naučené hypotézy se udržuje blízko přesnosti hypotézy, klasifikující instance náhodně); jinak řečeno algoritmus je směrován k tomu, aby v příštím kroku generoval slabou hypotézu, která bude v této „těžké části“ prostoru instancí dělat méně chyb. Výsledná hypotéza H přiřazuje instanci $\boldsymbol{x}$ váženou většinovou klasifikaci.

Jiným příkladem obalovací metody výběru příkladů je tzv. „technika oken“ (viz [28]); vznikla při experimentálním studiu vytváření rozhodovacích stromů a byla navržena jako způsob redukce času, potřebného pro sestrojení rozhodovacího stromu na velké tréninkové množině. Metoda vybere náhodně vzorek tréninkových dat, vytvoří pomocí něj (počáteční) rozhodovací strom a použije ho ke klasifikaci všech zbývajících tréninkových příkladů. Z nesprávně klasifikovaných příkladů vybere opět náhodně jiný vzorek, přidá ho k původnímu a (pomocí takto rozšířeného vzorku) vytvoří nový rozhodovací strom. Proces pokračuje, dokud nejsou správně klasifikovány všechny tréninkové příklady. Tento postup vedl k podstatné redukci času potřebného k vytvoření rozhodovacího stromu na velkém souboru šachových koncovek.

Lze si představit i filtrační metody výběru příkladů (například čištění tréninkových dat odstraněním příkladů, které se liší pouze klasifikací – jsou „nekonzistentní“), obvyklejší ale jsou vnořovací a obalovací metody.

2.2. Výběr z neohodnocených příkladů.

Může být užitečný v případě, že neohodnocených instancí je velmi mnoho (jsou snadno dosažitelné), ale získání klasifikace je náročné. Jedním ze základních postupů, který můžeme vnořit do induktivních algoritmů pracujících s množinami hypotéz konzistentních s tréninkovými příklady, je tzv. „query by committee“ (viz [32]).Pro neohodnocenou instanci vybereme náhodně dvě (konzistentní) hypotézy; dávají-li této instanci různé klasifikace, požádáme cílovou funkci o „správné“ohodnocení. Tento postup je založen na představě, že příklady, které dostanou od vybraných hypotéz různé predikce jsou cennější (nesou více informace) než ty, které většina hypotéz klasifikuje stejné. Řada studií se také týkala algoritmů, které samy generují příklady, na nichž se učí. Obecnou technikou, užívanou algoritmy tohoto druhu (někdy se v této souvislosti užívá termín „membership

query“) je vzít známý příklad, postupně měnit hodnoty atributů a sledovat efekt těchto změn na jeho klasifikaci (například vezmeme dva příklady s různým ohodnocením a „procházíme“ je současně tak dlouho, až najdeme místo způsobující změnu klasifikace; takovýto postup souvisí s určováním relevantních atributů, kterému jsme se podrobně věnovali v první kapitole). Jinou obecnou metodou je efektivní navrhování experimentů (příkladů), schopných rozlišit mezi soutěžícími hypotézami; tím se eliminují některé hypotézy a omezuje se složitost učení.

Většina pozornosti byla při výběru neohodnocených příkladů věnována vnořovacím metodám.

Literatura.

[1] Almuallim, H., Dietterich, T.G. : Learning with many irrelevant features; in Proceedings AAAI-91, Anahaim, CA, (MIT Press, Cambridge, MA, 1991), 547 – 552

[2] Almuallim, H., Dietterich, T.G. : Learning Boolean concepts in the presence of many irrelevant features; Artrficial Intelligence 69 (1994), 279 – 306

[3] Blum, A.L., Langley, P. : Selection of relevant features and examples in machine learning; Artificial Intelligence 97 (1997) 245 – 271

[4] Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K. : Occam`s razor; Inform. Process. Lett. 24 (1987), 377 – 380

[5] Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K. : Lernability and the Vapnik-Chervonenkis dimension; Journal of the Association for Computing Machinery, 1989, 36 (4), 929 – 965

[6] Cardie, C. : Using decision trees to improve case-based learning; in Proceedings 10th Internatioal Conference on Machine Learning; Amherst, MA (M.Kaufmann, Los Altos, CA, 1993), 25 – 32

[7] Clark, P., Niblett, T. : The CN2 induction algorithm; Machine Learning 3 (1989), 261 – 284

[8] Cover, T.M., Hart, P.E. : Nearest neighbor pattern classification; IEEE Trans. Inform. Theory 13 (1967), 21 – 27

[9] Dhagat, A., Hellerstein, L. : PAC learning with irrelevant attribures; in Proceedings IEEE Symposium on Foundation of Computer Science (IEEE, 1994), 64 – 74

[10] Freund, Y., Schapire, R.E. : A decision-theoretic generalization of on-line learning and an application to boosting; Journal of Computer and System Sciences, 55 (1), 1977, 119 – 139

[11] Garey, M., Johnson, D. : Computers and Intractability : A Guide to the Theory of NP-Completeness; W.H.Freeman, San Francisco, CA, 1979

[12] Haussler, D., : Quantifying the inductive bias in concept learning; in Proceedings AAAI-86, Philadelphia, PA (AAAI Press, 1986), 485 – 489

[13] John, G., Kohavi, R., Pfleger, K. : Irrelevant features and the subset selection problem; in Proceedings 11th International Conference on Machine Learning, New Brunswick, NJ (M. Kaufmann, Los Altos, CA, 1994), 121 – 129

[14] Johnson, D.S. : Approximation algorithms for combinatorial problems; J. Comput. System Sci. 9 (1974), 256 – 278

[15] Kearns, M.J., Vazirani, U.V. : An Introduction to Computational Learning Theory; MIT Press, Cambridge, MA, 1994

[16] Kira, K., Rendell, L.A. : The feature selection problem : Tradicional methods and a new algorithm; in Proceedings AAAI-92, San Jose, CA (MIT Press, Cambridge, MA, 1992), 129 – 134

[17] Kira, K., Rendell, L.A. : A practical approach to feature selection; in Proceedings 9th International Conference on Machine Learning, Aberdeen, Scotland (M. Kaufmann, Los Altos, CA, 1992), 249 – 256

[18] Kivinen, J., Warmuth, M.K., Auer, P. : The Perceptron algorithm versus Winnow : linear versus logarithmic mistake bounds when few input variables are relevant; Artificial Intelligence 97 (1997), 325 – 343

[19] Kohavi, R., John, G.H. : Wrappers for feature subset selection; Artificial Intelligence 97 (1997), 273 – 324

[20] Kononenko, I. : Estimating attributes : analysis and extensions of Relief; in Bergadano, F, De Raedt, L. (eds) : Proceedings 7th European Conference on Machine Learning, 1994

[21] Langley, P., Iba, W. : Average-case analysis of a nearest neighbor algorithm; in Proceedings IJCAI-93, Chambery, France (1993), 889 – 894

[22] Langley, P., Sage, S. : Oblivious decision trees and abstract cases; in Working Notes of the AAAI-94 Workshop on Case-Based Reasoning, Seattle, WA (AAAI Press, 1994), 113 – 117

[23] Littlestone, N. : Learning quickly when irrelevant attributes abound : a new linear treshold algorithm; Machine Learning 2 (1988), 285 – 381

[24] Michalski, R. S. : Pattern recognition as rule-guided inductive inference; IEEE Trans. Pattern Anal. Machine Intell. 2 (1980), 349 – 361

[25] Minski, M., Papert, S. : Perceptrons : An Introduction to Computational Geometry (MIT Press, Cambridge, MA, 1969)

[26] Mitchell, T. M. : Generalization as search; Artificial Intelligence 18 (1982), 203 – 226
[27] Pagallo, G., Haussler, D. : Boolean feature discovery in empirical learning; Machine Learning 5 (1990), 71 – 99
[28] Quinlan, J. R. : Learning efficient classification procedures and their application to chess end games; in Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds) : Machine Learning : An Artificial Intelligence Approach, Vol 1, M. Kaufmann, San Mateo, CA, 1983
[29] Quinlan, J.R. : C 4.5 : Programs for Machine Learning; M. Kaufmann, San Mateo, CA, 1993
[30] Rivest, R.L. : Learning decision lists; Machine Learning 2 (3), 1987, 229 – 246
[31] Rosenblatt, F. : The Perceptron : a probabilistic model for information storage and organization in the brain; Psych. Rev. 65 (1958), 386 – 407; přetištěno v : Neurocomputing (MIT Press, Cambridge, MA, 1988)
[32] Seung, H.S., Opper, M., Sompolinski. H. : Query by committee; Proceedings 5th Annual Workshop on Computational Learning Theory, Pittsburgh, PA (ACM Press, New York, 1992), 287 – 294
[33] Schapire, R.E. : A brief Introduction to boosting; in Proceedings IJCAI-99, Stockholm, Sweeden, 1999, 1401 – 1406
[34] Vere, S.A. : Induction of concepts in predicate calculus; in Proceedings IJCAI – 75, Tbilisi, Georgia (M. Kaufmann, San Mateo, CA, 1975), 281 – 287
[35] Winston, P.H. : Learning structural description from examples; in Winston, P.H. (ed) : The Psychology of Computer Vision (McGraw-Hill, New York, 1975)