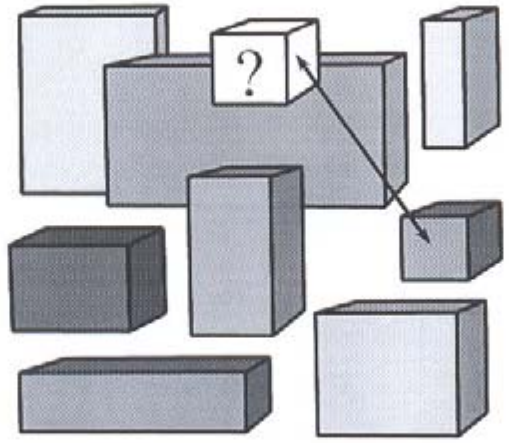
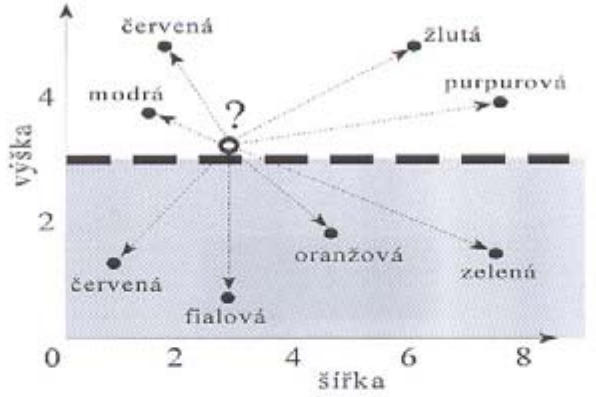
Učení zaznamenáváním případů

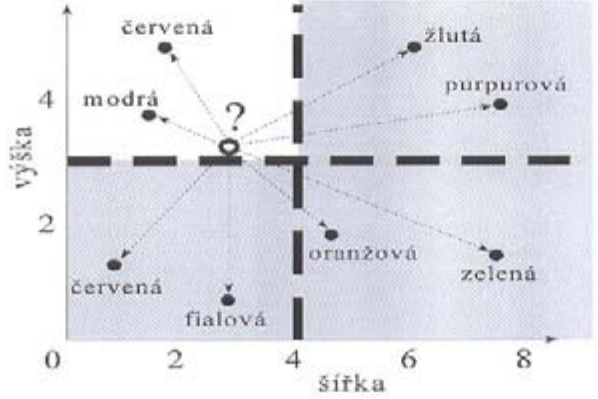
- Lze použít v případech, kdy není možné vytvořit dobrý model.
- ***Konsistenční heuristika***: používá se k odhadování neznámých vlastností nových případů porovnáním s případy zaznamenanými:

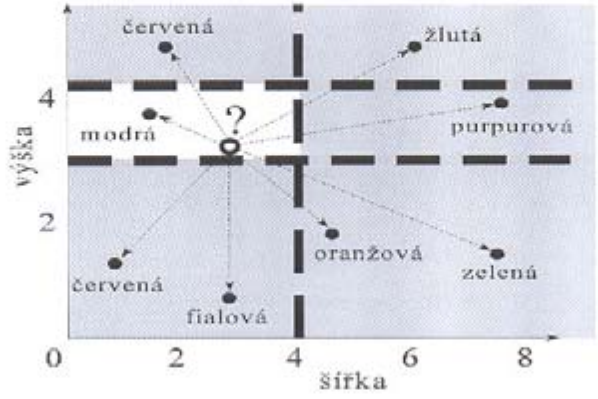
  Kdykoliv je zapotřebí odhadnout vlastnost nějakého objektu a není k dispozici nic, než soubor zaznamenaných referenčních případů, postupuje se tak, že se najde nejpodobnější případ, jehož vlastnosti jsou známy. O neznámé hodnotě atributu se pak předpokládá, že je stejná jako u podobného, již zaznamenaného případu. Podobnost se určuje vhodně zvolenou metrikou.

- Příklad: v paměti jsou uloženy údaje o osmi kvádrech různé velikosti a barvy. Jsou-li u nového, dosud nezaznamenaného kvádru (případu) známy rozměry a není známa barva, pak — není-li k dispozici jiné vodítko — se určí stejná barva jako u již zaznamenaného kvádru rozměrově nejpodobnějšího.



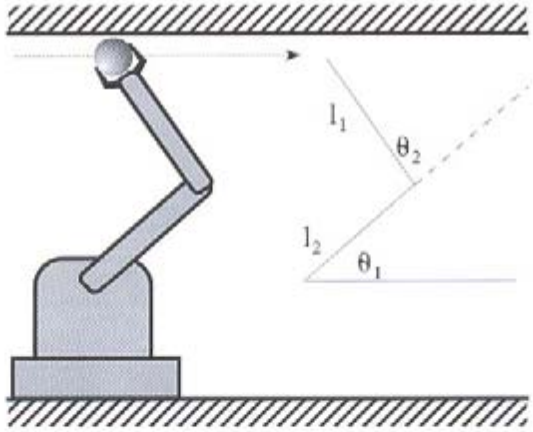






Kvádr neznámé barvy je klasifikován jako *modrý*, neboť je modrému vzorku nejblíže svou výškou a šířkou.

- ***Konsistenční heuristika***: umožňuje řešit obtížné dynamické problémy:

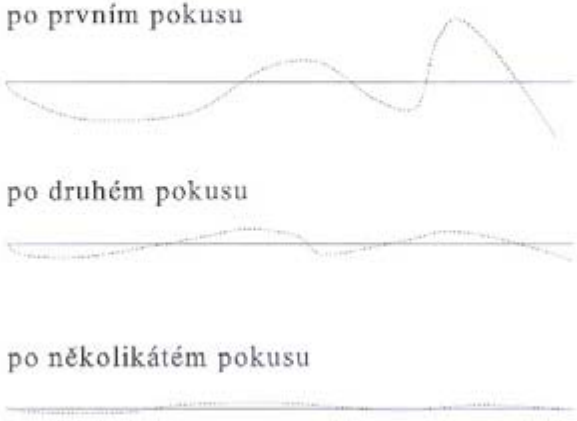


Model ramene robota, přenášejícího míček, lze sestavit pomocí diferenciálních rovnic, kde točivé momenty motorů v kloubech jsou funkcí úhlů θ a délek ramen l. Problém reálného světa je však zcela jiný: požadované momenty závisejí ve skutečnosti na rychlostech (a jejich mocninách a součinech) a zrychleních pohybu ramen, což zahrnuje Coriolisovy a dostředivé síly a dále proměnné vzájemně závislé setrvačné momenty.

I kdyby se podařilo zachytit v rovnicích vztahy mezi všemi atributy, výsledky s řízením reálného ramene robota nebudou uspokojivé a neposkytnou vysvětlení, jak je schopno biologické rameno dobře vrhat míček — existuje příliš mnoho faktorů jež je nutno uvažovat a příliš mnoho veličin, jež je nutno přesně měřit.

- Metoda ***nejbližšího souseda*** umožňuje praktická řešení podobných problémů. Lze např. nechat více-méně náhodně konat rameno pokusné pohyby a zaznamenávat do tabulky hodnoty momentů, úhlů, rychlostí atp. Chceme-li, aby byl míček dopraven po konkrétní trajektorii, pak ji rozdělíme na malé úseky a na tabulku budeme hledět jako na mnoharozměrný prostor různých řešení. Pro každou pozici blízko požadované trajektorie zjistíme, které hodnoty parametrů (úhly, rychlosti, zrychlení ...) jí odpovídají a interpolací mezi nimi najdeme potřebné momenty odpovídající žádané dráze.

Tabulka ovšem nebude často dostatečně hustě vyplněná (tj. prostor řešení bude vyplněn řídce příslušnými body). Praktickým řešením je nechat udělat první pokus, který bude nejspíš špatný. Po několika dalších pokusech (a tím i zápisu nových údajů do tabulky) začne docházet ke zlepšování — nové vstupní údaje budou lepší než staré a pohyb bude nakonec uspokojivý:



- ***Hledání nejbližšího souseda*** — lze buď sekvenčně nebo paralelně.

★ **Sekvenční:** spočítá se vzdálenost od ostatních případů a zvolí se nejbližší případ. Pro n ostatních případů existuje n výpočtů vzdáleností a $n-1$ porovnání těchto hodnot. Přímočarý postup je použitelný pro malá n (např. 10), ale ne např. pro $n=10^6$.

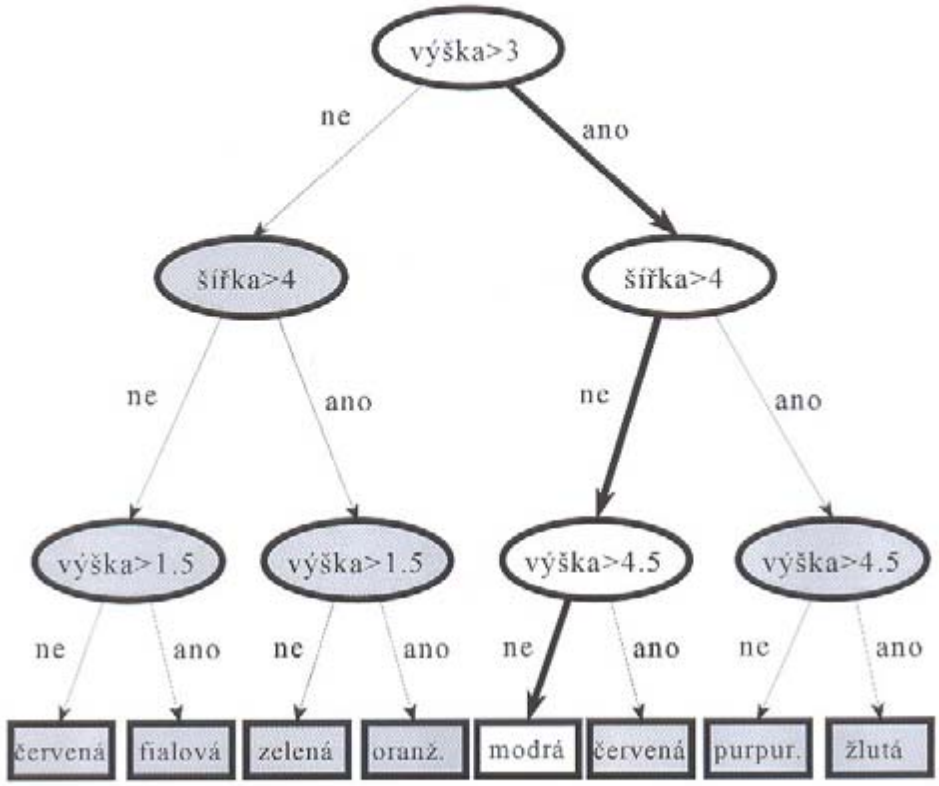
Problém velkého počtu operací lze vyřešit použitím *rozhodovacích stromů*, kdy je počet operací úměrný $log_2 n$ místo n. Pro příklad kvádrů lze postupovat takto: kvádry se rozdělí na dvě množiny např. podle výšky (menší než nějaká hodnota a větší) tak, aby obě části měly stejný počet prvků. Dále se každá tato množina podobně rozdělí podle šířky, pak zase podle výšky atd. Výsledkem je tzv. ***k-d*** strom (k-dimensional tree):

k-d strom je reprezentace rozhodovacího stromu, v němž:

- soubor možných odpovědí se skládá z bodů, z nichž jeden může být nejbližším sousedem daného bodu;
- každý test specifikuje souřadnici, práh a neutrální zónu v okolí prahu, která neobsahuje žádné body;
- každý test dělí soubor bodů na dvě části v souladu s tím, na které straně prahu každý bod leží.

Nalezení *nejbližšího* kvádru je tedy pouze záležitostí sledování příslušné cesty v rozhodovacím stromu, která reflektuje způsob, jak jsou objekty rozdělovány do podmnožin.

Obecně platí, že má-li rozhodovací strom faktor rozvětvení 2 a hloubku d, pak bude mít 2^d listů. Má-li být identifikováno n objektů, musí být splněno, že $\mathbf{2^d \geq n}$. Logaritmováním obou stran dostáváme, že počet porovnání $\approx log_2 n$.



★ **Paralelní:** Kdybychom měli k dispozici masivně-paralelní počítač, kde pro každý případ by byl jeden procesor, pak není zapotřebí uvedené důmyslné hledání. Každé měření vzdálenosti by šlo provést paralelně. Samozřejmě všechny výsledky musí být nějak porovnány, aby byla stanovena nejmenší vzdálenost atributu s neznámou hodnotou. Bylo by např. možné používat sousední procesory k porovnání jejich výsledků. Každé takové dvouprocesorové minimum by opět bylo porovnáno se sousedním dvouprocesorovým minimem. Tento postup by nakonec dal globální minimum po počtu sekvenčních kroků řádu $log_2 n$, kde n je počet porovnávaných vzdáleností. Existují ovšem lepší způsoby jak najít minimální vzdálenost v konstatním čase na paralelním počítači.

Algoritmus rozdělování případů do množin:

- je-li jen jeden případ, STOP;
- jde-li o první dělení, zvol pro srovnávání vertikální osu, jinak zvol osu různou od osy použité na nejbližší vyšší úrovni;
- vzhledem k ose dělení, najdi polohu mezi dvěma prostředními objekty a nazvi tuto pozici *práh*; vytvoř test rozhodovacího stromu jenž porovnává neznámé objekty vůči prahu; zaznamenej pozici obou prostředních objektů na ose srovnání a nazvi tyto pozice *spodní* a *horní* hranice;
- rozděl všechny objekty na dva podsoubory podle toho, na které straně střední pozice leží;
- rozděl objekty v každé z obou množin a vytvoř tak podstrom pro každou z nich za použití výše uvedené procedury.

Algoritmus *k-d* procedury pro hledání:

- urči, zda existuje pouze jediný element v uvažovaném souboru;
- pokud ano, zaznamenej jej, jinak
- porovnej klasifikovaný objekt na ose porovnání vůči současnému prahu uzlu; výsledek určuje množinu podobných objektů;
- použitím této procedury najdi v určené množině nejbližšího souseda;
- urči, zda vzdálenost k nejbližšímu sousedovi v množině ≤ vzdálenost ke druhé hranici množiny na ose porovnávání;
  - pokud ano, zaznamenej nejbližšího souseda v této množině, jinak
  - otestuj druhou množinu touto procedurou; jako výsledek vrať bližšího z obou blízkých sousedů v obou množinách.

Metoda k-nejbližšího souseda (k-Nearest Neighbor) k-NN

Metoda k-NN předpokládá, že všechny příklady (instance) odpovídají bodům v n-rozměrném prostoru $\mathbf{R}^n$. Nejbližší soused nějaké instance je určen pomocí termínů standardní euklejdovské vzdálenosti.

Nechť je libovolná instance x popsána vektorem

$$\langle a_1(x), a_2(x), \ldots, a_n(x) \rangle$$

kde $a_r(x)$ označuje hodnotu r-tého atributu instance x. Pak lze definovat vzdálenost mezi dvěma instancemi x_i a x_j:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} \left(a_r(x_i) - a_r(x_j)\right)^2}$$

Cílová funkce, kterou se algoritmus k-NN naučí, může být jak reálná (s reálnými hodnotami), tak s diskrétními hodnotami.

Pro $f: \mathbf{R}^n \to V$ (kde V je konečná množina $\{v_1, \ldots, v_s\}$)

Trénovací algoritmus

Pro každý trénovací příklad $\langle x, f(x) \rangle$, přidej jej do seznamu trénovací_příklady.

Klasifikační algoritmus

- Je dána instance x_q ke klasifikaci:
  - Nechť $x_1, \ldots, x_k$ označuje k instancí z množiny trénovací příklady, které jsou nejbližší k x_q;
  - Urči

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

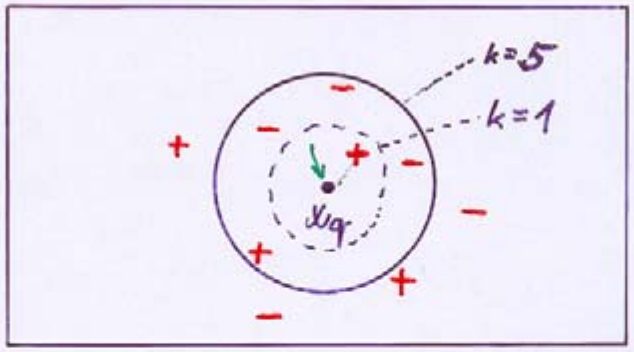
kde $\delta(a,b) = 1$ když $a = b$ a $\delta(a,b) = 0$ jinak.

Pozn.: $\underset{x \in X}{\operatorname{argmax}}\, f(x)$ vrací hodnotu x, která maximalizuje $f(x)$. Např. $\underset{x \in \{1,2,-3\}}{\operatorname{argmax}}\, x^2 = -3$

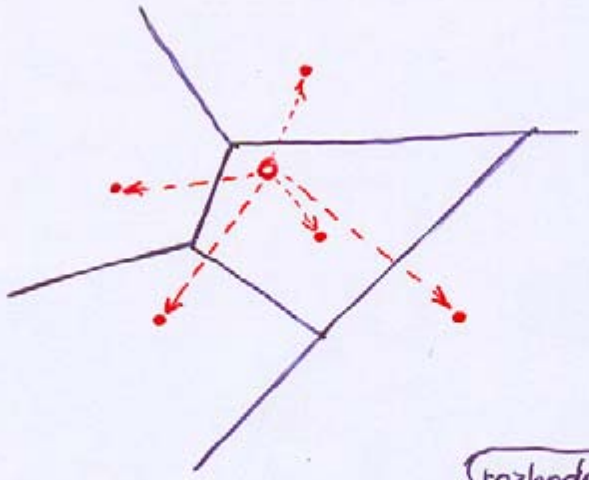
$\hat{f}(x)$ označuje funkci, která aproximuje $f(x)$.

Algoritmus vrací hodnotu $\hat{f}(x_q)$ jako svůj odhad $f(x_q)$, což je prostě nejčastější hodnota f mezi k trénovacími příklady blízkými x_q.

Pro $k = 1$ vrací 1-NN jako $\hat{f}(x_q)$ hodnotu $f(x_i)$, kde x_i je trénovací instance nejbližší k x_q.
Pro $k > 1$ vrací k-NN nejčastěji se vyskytující hodnotu mezi k nejbližšími instancemi.



Je dán soubor pozitivních + a negativních − příkladů (trénovací množina). Neznámý případ předložený ke klasifikaci je x_q. 1-NN algoritmus klasifikuje x_q pozitivně (čárkovaný kruh), zatímco 5-NN negativně (plný kruh).



rozhodovací

Voroného diagram (Voronoi d.) – prostor indukovaný 1-NN algoritmem (konvexní polygon okolo každé trénovací instance indikuje oblast nejblíže tomuto bodu).

Aproximace funkcí na spojitém universu:
$f: \mathbb{R}^n \to \mathbb{R}$

Algoritmus k-NN lze snadno adaptovat. Místo zjišťování nejčastější hodnoty se spočítá střední hodnota k-nejbližších instancí; poslední řádek algoritmu se tedy změní na:

$$\hat{f}(x_q) = \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

---

k-NN váhovaný vzdálenostmi sousedů

Jedním nabízejícím se zlepšením uvedeného k-NN je váhování přínosů jednotlivých sousedů pomocí vzdálenosti (vychází se z myšlenky, že bližší soused přispívá k výsledku více):

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \, \delta(v, f(x))$$

$$w_i = \frac{1}{d(x_q, x_i)^2}$$

Pro případ, že by x_q bylo přesně rovno x_i a tudíž $d(x_q, x_i) = 0$, se použije $\hat{f}(x_q) \leftarrow f(x_i)$. Pro $\mathbb{R}$:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

$\forall x_i, f(x_i) = c \Rightarrow \hat{f}(x_q) = c$

$\sum_{i=1}^{k} w_i$ → konstanta působící jako normalizátor

Vzdáleností váhovaný k-NN je vysoce efektivní induktivní inferenční metoda používaná pro mnoho praktických problémů. Je odolná k výskytu šumu v trénovacích datech a velmi efektivní, pokud je k dispozici dostatek trénovacích dat.

Vzdálenost mezi instancemi se počítá pomocí všech atributů (všech os euklejdovského prostoru). Tím se liší od metod založených na pravidlech a rozhodovacích stromech, kde se pro vytváření hypotézy používá jen podmnožina.

Citlivost na irelevantní atributy: např. z 20 pouze 2 atributy jsou relevantní pro klasifikaci. Dvě instance s totožnými hodnotami obou relevantních atributů, avšak s různými hodnotami ostatních, mohou být velmi vzdáleny a proto klasifikovány chybně.

ALGORITMY IBL

(Instance-Based Learning — učení založené na instancích)

- Algoritmy IBL jsou odvozeny z metody klasifikace pomocí nejbližšího souseda *k-d*. IBL se velmi podobá *k-d* v tom, že také ukládá a používá pouze vybrané instance ke generování předpovědi klasifikace. IBL patří k inkrementálním metodám s učitelem.

- Pokusy ukázaly, že u nejbližšího souseda lze snížit požadavky na paměť s malými ztrátami přesnosti klasifikace, ale tuto úsporu nelze předpovědět. Velikost předpokládaných paměťových požadavků je polynomiálně úměrná velikosti cílových hranic v instančním prostoru. Algoritmy nejbližšího souseda dále nezlepšují přesnost klasifikace s tím, jak přicházejí nové instance a tím jsou ignorovány některé problémy reálného světa, např. šum — proto jsou takové algoritmy málo robustní (odolné).

- IBL algoritmy jsou v podstatě tvořeny čtyřmi základními skupinami:

  1. ***IB1*** — nejjednodušší, ukládají všechny tréninkové instance a jsou proto velmi náročné na paměť.
  2. ***IB2*** — modifikovaný IB1, jenž ukládá pouze chybně určené tréninkové instance (na začátku ovšem nejsou určené žádné). Liší se tedy ve fázi učení. Pokud je nově předložený příklad klasifikován správně, je ihned zapomenut. Jsou znatelně sníženy požadavky na paměť na úkor velmi malého snížení přesnosti klasifikace. IB2 je ale značně citlivý na šum, protože nesprávně klasifikované instance jsou především ty, jež jsou na rozhraní více možností (blízko hranic konceptů), dále výjimky a konečně data s šumem.
  3. ***IB3*** — obecně odolnost vůči šumu rozhoduje o robustnosti metod učení v praxi. pro rozhodovací stromy byly vyvinuty metody větvení založené na statistických testech k tolerování zašuměných dat. Podobné metody existují pro rozhodovací pravidla. IB3 je IBL rozšířený o významový test, který indikuje a rozlišuje zašuměné instance. Předpokládá se, že již uložené instance jsou spolehlivými klasifikátory před tím, než jsou použity pro následné klasifikační úlohy. Je-li přítomen šum, přesnost klasifikace se zlepšuje za současného snížení požadavků na paměť.

     V principu se používají dva prahy důvěry:
     — práh pro akceptování nebo
     — práh pro odmítnutí.

     Významový test rozhoduje o tom, zda instance je přijatelná nebo zašuměná. Přijatelná je tehdy, když její přesnost klasifikace je statisticky významně větší než její pozorovaná frekvence ve třídě. V opačném případě je instance vypuštěna. Zašuměné instance mají malou přesnost klasifikace, protože jejich nejbližší sousedé v instančním prostoru mohou mít jinou klasifikaci. Uvedená metoda je tolerantní vůči šumu, ale nedokáže rozlišit zašuměná data od výjimek — každá výjimka vypadá jako instance obsahující šum.

     Rozdíl IB3 oproti IB2: a) Udržuje klasifikační záznamy všech uložených instancí, tj. počet správných a nesprávných pokusů o klasifikaci. b) Akceptovány pro budoucí klasifikace jsou jen instance s dobrým klasifikačním záznamem. c) Algoritmus odolný šumu ničí ty uložené instance, které se jeví jako zašuměné (tj. ty, jejichž klasifikace je špatná po několika klasifikačních pokusech).

4. ***IB4*** — je rozšířením IB3 v tom smyslu, že nepředpokládá stejnou závažnost všech atributů pro predikci. IB4 je úspěšný tam, kde je instance popsána mnoha atributy. Zatímco u rozhodovacích stromů je význam atributu stanoven pomocí entropie (informačního zisku), IBL využívají jinou strategii:

   Závažnost atributu (jeho *relevance*) je dána nastavením váhy pro daný atribut. Při učení jsou zaznamenány vyšší váhy atributům, které se podílejí na správné klasifikaci. Každý cílový koncept má unikátní množinu vah atributů. IB4 nepředpokládá vyčerpávající a nepřekrývající se koncepty. Pro jednu instanci uvažuje jednu, žádnou či více cílových hodnot. Pro rozhodnutí, která z možností je správná, používá funkci *Estimace* (odhadu členství). Závažnější prediktory jsou zvýhodňovány, irelevantní zeslabovány. Každá váha atributu je aktualizována po každém pokusu predikce během fáze učení.

- ***Princip IBL:*** Tyto algoritmy se hodí k řešení úloh, kde zdrojem příkladů (instancí) je vnější prostředí. Algoritmus pasivně přijímá instance vnějšího procesu. Každá instance je reprezentována množinou hodnot atributů. Instance, jež je popsána *n* atributy, je uložena v *n*-dimenzionálním instančním prostoru. Atributy jsou definovány nad množinou numerických nebo symbolických hodnot. IBL může tolerovat chybějící hodnoty atributů.

  Jeden z atributů, použitých pro popis instancí, je použit jako tzv. ***cílový atribut***, ostatní jsou tzv. ***prediktory***. Základní vlastností algoritmů je schopnost naučit se předpovídat hodnotu cílového atributu u instance, jejíž cílová hodnota chybí.

  Množina všech instancí v instančním prostoru, které mají stejnou hodnotu cílového atributu, se nazývá ***třída*** (též kategorie). Tzv. ***koncept*** je cílový popis kategorie. IBL jsou předkládány sekvence tréninkových instancí a koncepty k naučení pro každou hodnotu cílového atributu (koncepty jsou funkce, rozdělující instance do tříd). ***Cílem je přesně určit cílové atributy***.

- ***IBL se skládá ze čtyř základních funkcí:*** normalizace, podobnost, predikce, aktualizace paměti.

  - ***normalizace*** — normalizuje číselné hodnoty predikčních atributů. Při předzpracování instancí se udržují a aktualizují celkové informace týkající se hodnot každého číselného atributu během tréninkového procesu. Tato informace se používá pro normalizaci, která upravuje velikost každé numerické hodnoty atributu. Pomocí normalizace se dosahuje toho, že každý atribut má stejnou závažnost. K normalizaci lze např. použít jednoduchý lineární vztah využívající nejnižší a nejvyšší hodnotu atributu:

$$x_n = \frac{x - X_{\min}}{X_{\max} - X_{\min}}$$

kde x_n je nová (normalizovaná) hodnota atributu, x je načtená hodnota (skutečná) atributu, X_{min} je minimální hodnota atributu a X_{max} je maximální hodnota atributu.

  - ***podobnost*** — funkce podobnosti pracuje s částečným konceptem a upravenou instancí. Výsledkem je numericky vyjádřená podobnost nové instance s každou instancí v částečném konceptu. Jednoduchou funkcí může např. být inverze Euklidovy vzdálenosti dvou bodů x a y v instančním prostoru (u symbolických atributů je vzdálenost nulová při identické shodě, jinak je rovna 1):

$$Podobnost\,(x,y) = \frac{1}{\sqrt{\sum_{i \in P} Vzdálenost\_atributů\,(x_i, y_i)}}$$

kde funkce $Vzdálenost\_atributů = (x_i - y_i)^2$ pro numerické hodnoty a $x_i \neq y_i$ pro symbolické hodnoty. Chybí-li obě hodnoty, pak je výsledek 1, chybí-li y_i resp. x_i, pak je výsledek $max(x_i\text{-}0,\ 1\text{-}x_i)$, resp. $max(y_i\text{-}0,\ 1\text{-}y_i)$. P v uvedeném vztahu je počet predikčních atributů.

  - ***predikce*** — vstupem jsou vypočtené hodnoty podobnosti a výstupem predikce hodnot cílových atributů instancí. Jednoduchou funkcí predikce může být např. metoda nejbližšího souseda (pro symbolické i numerické hodnoty). Výstupem jsou hodnoty cílových atributů instancí s nejvyšší podobností do částečného konceptu. Mají-li aspoň dvě instance shodnou maximální podobnost, může být proveden náhodný výběr jedné z nich (a ta bude predikovat cílovou hodnotu). další možností je použití tzv. *většinového hlasování*, především pro symbolické hodnoty. Předpovězená hodnota je pak ta, která se stane nejčastější mezi k-nejpravděpodobnějšími tréninkovými instancemi.

- ***aktualizace paměti*** — částečný koncept je aktualizován poté, co byla vypočtena podobnost, resp. váha každé tréninkové instance. Částečný koncept se skládá z množiny instancí a z přidružené informace ve vztahu k prospěšnosti uložených instancí a atributů, které je popisují. Je možné, že tato funkce bude poskytovat další užitečné informace (např. parametry pro predikce). Nejjednodušší funkcí aktualizace je jednoduché ukládání všech tréninkových instancí do částečného konceptu. Složitější funkcí je např. vypouštění instancí, které se jeví jako zašuměné, z konceptu.