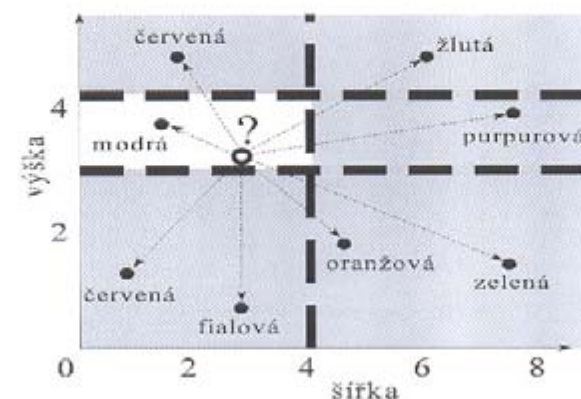
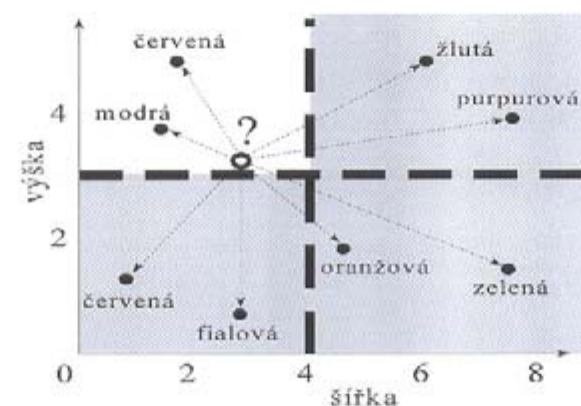
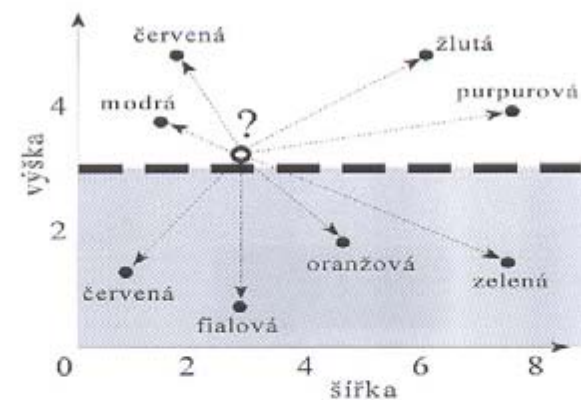
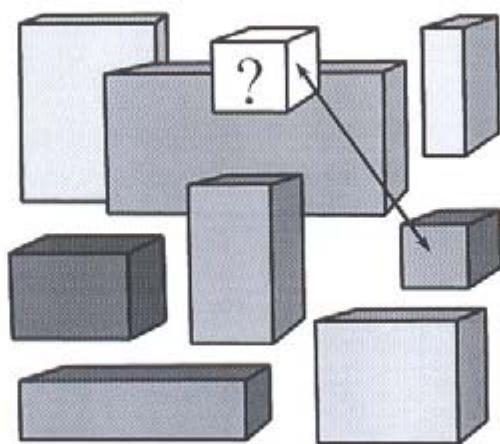


UČENÍ ZAZNAMENÁVÁNÍM PŘÍPADŮ

- 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.

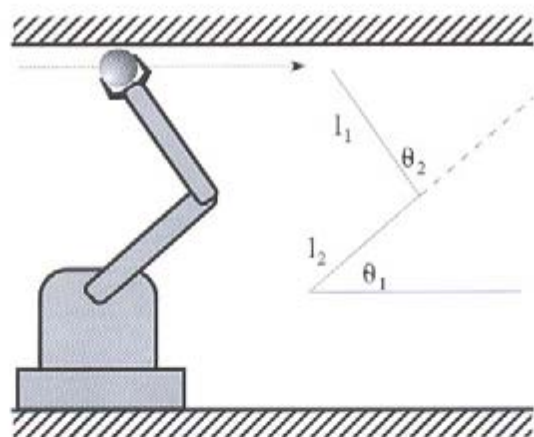
- Příklad: v paměti jsou uloženy údaje o osmi kvádrech různé velikosti a barvy. Jsou-li u nového, dosud nezaznamenaného kvádru (případu) známy rozměry a není známa barva, pak — není-li k dispozici jiné vodítko — se určí stejná barva jako u již zaznamenaného kvádru rozměrově nejpodobnějšího.





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

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



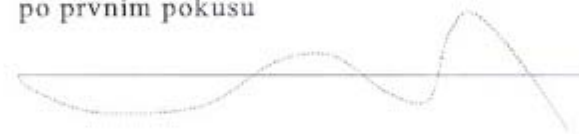
Model ramene robota, přenášejícího míček, lze sestavit pomocí diferenciálních rovnic, kde točivé momenty motorů v kloubech jsou funkcí úhlů θ a délek ramen l . Problém reálného světa je však zcela jiný: požadované momenty závisejí ve skutečnosti na rychlostech (a jejich mocninách a součinech) a zrychleních 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 mnohazměrný 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 řídkce 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:

po prvním pokusu



po druhém pokusu



po několikátém pokusu



- **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 $n=10^6$.

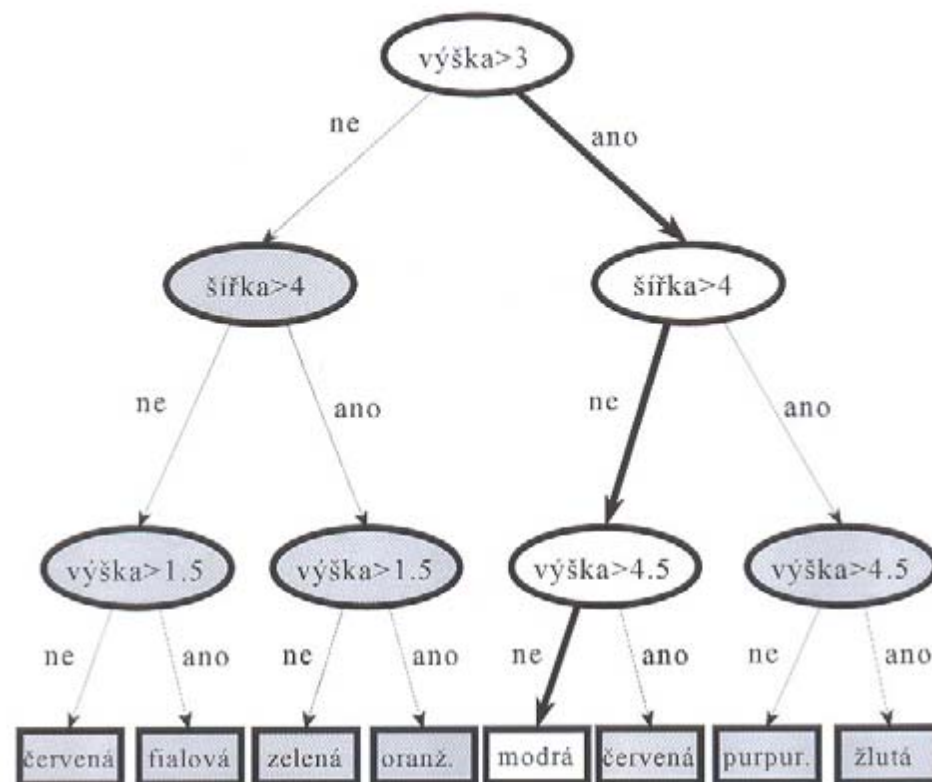
Problém velkého počtu operací lze vyřešit použitím *rozhodovacích stromů*, kdy je počet operací úměrný $\log_2 n$ místo n . Pro příklad kvádrů lze postupovat takto: kvádry se rozdělí na dvě množiny např. podle výšky (menší než nějaká hodnota a větší) tak, aby obě části měly stejný počet prvků. Dále se 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* kvádrů 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 $2^d \geq n$. 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 minimumem. 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 konstantním 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 \leq 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 \mathbb{R}^n . Nejbližší soused nějaké instance je určen pomocí termínu standardní eukleidovské vzdálenosti.

Nechť je libovolná instance x popsána vektorem

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

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

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

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

Pro $f: \mathbb{R}^n \rightarrow V$ (kde V je konečná množina $\{N_1, \dots, N_s\}$),

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, \dots, x_k označuje k instancí z množiny trénovací příklady, které jsou nejblíže k x_q ;
 - Urči

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

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

Pozn.: $\operatorname{argmax}_{x \in X} f(x)$ vrací hodnotu x , která

maximalizuje $f(x)$. Např. $\operatorname{argmax}_{x \in \{1, 2, -3\}} x^2 = -3$

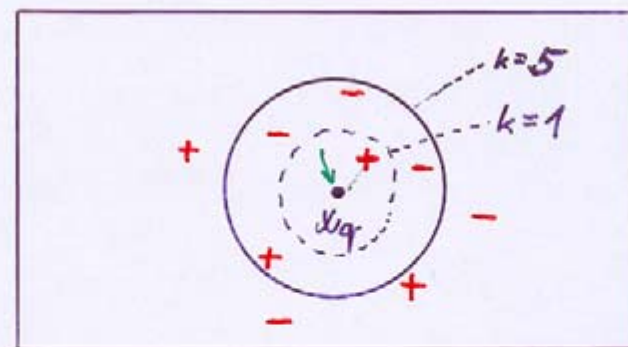
$\hat{f}(x)$ označuje funkci, která aproximuje $f(x)$.

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 .

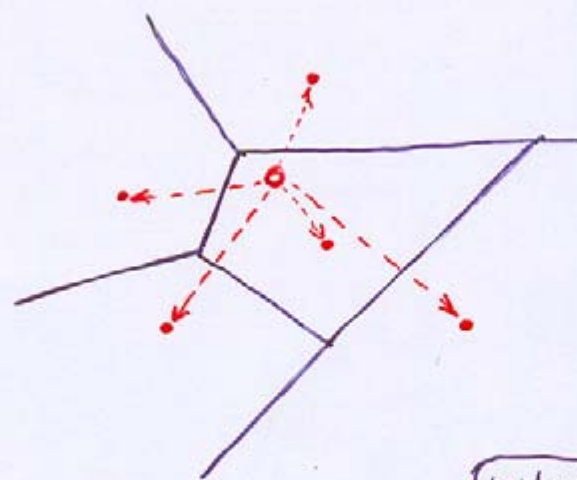
Pro $k=1$ vrací 1-NN jako $\hat{f}(x_q)$ hodnotu $f(x_i)$,

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). Nerávný 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 nejbliže tomuto bodu).

Aproximace funkce na spojitém universu:

$$f: \mathbb{R}^n \rightarrow \mathbb{R}$$

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):

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

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

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} :

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i} \quad \begin{array}{l} \forall x_i, f(x_i) = c \Rightarrow \\ \Rightarrow \hat{f}(x_q) = c \end{array}$$

→ konstanta působící jako normalizátor

Vzdáleností váhovaný k-NN je vysoce efektivní indukční 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 eukleidovského prostoru). Tím se liší od metod založených na pravidlech a rozhodovacích stromech, kde se pro vytrávení hypotézy používá jen podmnožina.

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