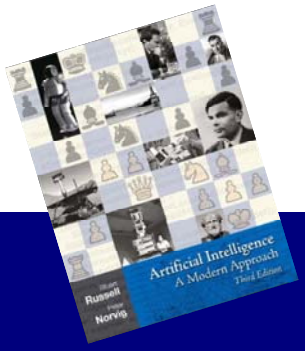


Umělá inteligence II

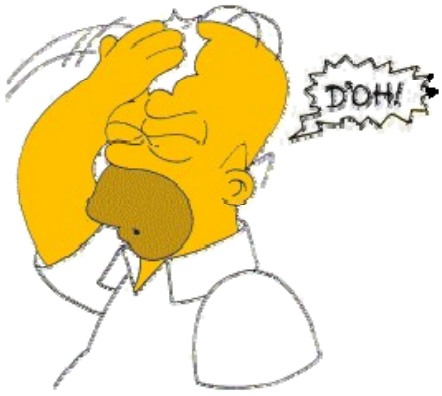
Roman Barták, KTIML

`roman.bartak@mff.cuni.cz`
`http://ktiml.mff.cuni.cz/~bartak`

9

Úvodem

- Dnes budeme **učit agenty**, jak zlepšit svůj výkon při řešení budoucích úloh na základě pozorování světa.
  - rozhodovací stromy
  - regresní metody
  - umělé neuronové sítě
  - neparametrické modely
  - support vector machines
  - boosting



Proč se učit?

Pokud můžeme chování agenta zlepšit, proč ho rovnou nenaprogramujeme lépe?

- těžko můžeme **předvídat všechny situace**, do kterých se agent může dostat
  - Můžeme agenta naprogramovat na průchod jedním bludištěm, ale projde všechna bludiště?
- těžko můžeme **předvídat všechny změny** světa v čase
  - pokud předvídáme cenu na burze, je potřeba se adaptovat na změnu situace
- často ani **nemusíme vědět**, jak řešení úkolu naprogramovat
  - každý umíme rozpoznávat obličeje blízkých, ale jak proces rozpoznávání naprogramovat?

Formy učení

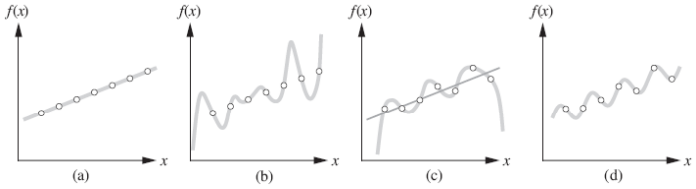
V principu můžeme zlepšovat každou část agenta.

Zlepšení a použité techniky záleží na zodpovězení otázek:

- Jakou **část agenta** zlepšujeme?
  - funkci užitku, efekty akcí, převod pozorování na akci, …
- Jak **reprezentujeme znalosti** a zlepšovanou část? Jaké **předchozí znalosti** agent má?
  - logická reprezentace, Bayesovské sítě
- Na základě jaké **odezvy** se agent učí?
  - **učení bez učitele** (unsupervised learning)
    - například hledání vzorů klusterováním
  - **zpětnovazební učení** (reinforced learning)
    - učení na základě odměn a trestů
  - **učení s učitelem** (supervised learning)
    - z dvojic vstup-výstup se učíme odpovídající funkci

Učení s učitelem

- Máme dána **tréninková data** vstup-výstup $(x_1,y_1),\ldots,(x_N,y_N)$, kde $y_i = f(x_i)$.
- Cílem je najít **hypotézu** h, která aproximuje funkci f.
  - hypotézy vybíráme z **prostoru hypotéz** (například lineární funkce)



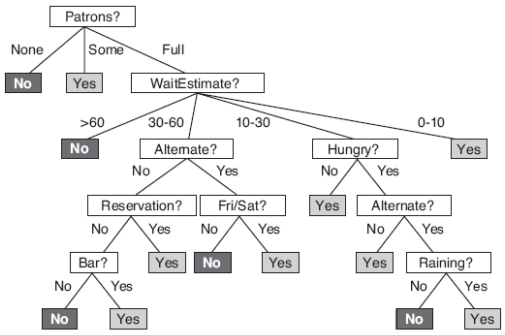
  - **hypotéza** je **konzistentní**, pokud $h(x_i) = y_i$
  - přesnost hypotézy měříme na **testovacích datech**
- Typy úloh
  - **klasifikace**: množina výstupů y_i je konečná (slunečno, zataženo, déšť)
  - **regrese**: výstupy jsou čísla (teplota vzduchu)

Ockhamova břitva

- Co když tréninkovým příkladům odpovídá **více hypotéz**?
- Obecně preferujeme **jednodušší hypotézu**.
  - Někdy může být lepší jednodušší hypotéza, která pouze aproximuje tréninková data, než složitější hypotéza, která přesně pokrývá tréninkové příklady (přeučení).
- Jaká hypotéza je jednodušší?
  - například lineární funkce je jednodušší než kvadratická
- Princip **Ockhamovy břitvy** – nejjednodušší vysvětlení je zpravidla správné.

Rozhodovací stromy

- **Rozhodovací strom** je jednoduchý způsob reprezentace funkce, která vektor vstupních atributů převádí na rozhodnutí – výstupní hodnotu.
  - rozhodnutí získáme posloupností testů na hodnoty vstupních atributů



- Uvažujme binární rozhodnutí (Booleovský rozhodovací strom)
  - pro n atributů je rozhodovací funkce popsatelná tabulkou s 2^n řádky
  - máme tedy 2^{2^n} různých funkcí
  - každou funkci můžeme triviálně popsat rozhodovacím stromem hloubky n

Jak najít „malý" konzistentní rozhodovací strom?

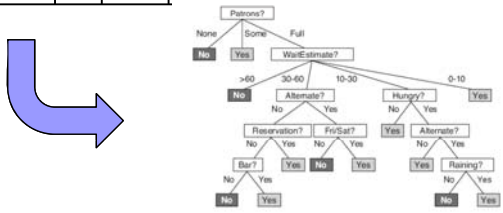
Rozhodovací stromy

řešená úloha

- Prostor hypotéz je tvořen rozhodovacími stromy a my hledáme ten, který odpovídá zadaným příkladům (konzistentní hypotéza).

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

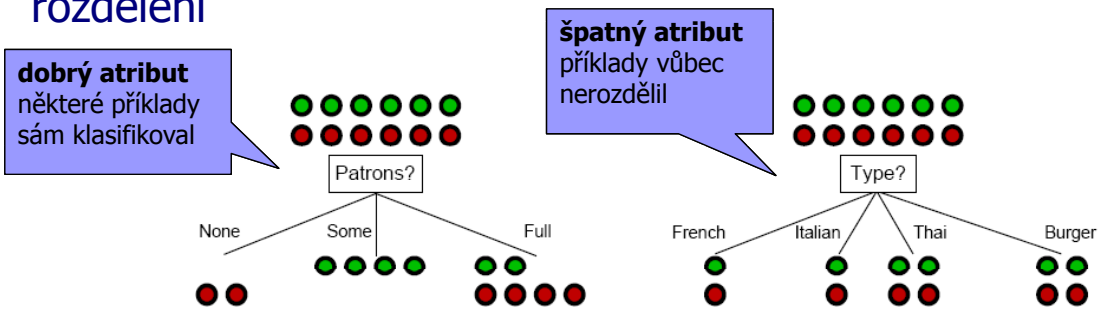
- příklady ve tvaru (**x**, y)
- pro jednoduchost budeme pracovat s Boolean výstupem
- příklad s restaurací (budeme čekat?)



Rozhodovací stromy

princip učení

- Strom budeme hledat metodou **rozděl a panuj**
  - vybereme nejvíce informativní atribut, který dáme do kořene
  - podle hodnot atributu rozdělíme příklady
  - pokud dostaneme příklady se stejným výstupem, jsme hotovi, jinak pokračujeme s dalším atributem
- Jak poznáme **nejvíce informativní atribut**?
  - je to ten, který do klasifikace přinese největší rozdělení



Rozhodovací stromy

algoritmus učení

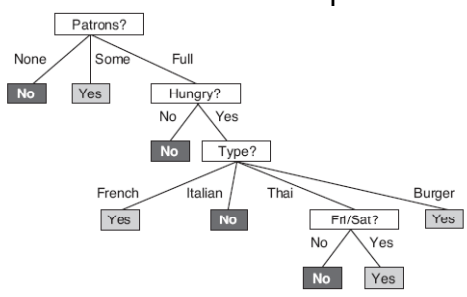
Algoritmus ID3

```
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
```

vrátí nejčastější výstup z příkladů rodiče

vrátí nejčastější výstup z příkladů pro rozdělení

vybere nejvíce informativní atribut



Volba testovacího atributu

Jaký atribut vybrat pro test v rozhodovacím stromu?

- Rozhodneme se podle **informačního zisku** atributu (větší je lepší), který je definován pomocí **entropie**.
- **Entropie** je míra neurčitosti náhodné proměnné
  - měří se v „bitech" informace, které získáme znalostí hodnoty proměnné
    - mince, kde vždy padá orel, má nulovou entropii
    - klasická mince má entropii 1
    - „mince" se čtyřmi stejně pravděpodobnými stranami má entropii 2

$H(V) = - \Sigma_k p(v_k) \log_2(p(v_k))$, v_k jsou hodnoty proměnné V

$B(q) = - q.\log_2 q - (1-q).\log_2(1-q)$ entropie Booleovské proměnné

$H(Goal) = B(p/(p+n))$ entropie sady p pozitivních a n negativních příkladů

- Atribut A rozdělí příklady dle hodnoty atributu
  - očekávaná entropie po testu atributu A

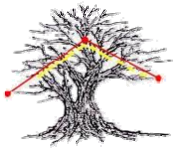
    $Remainder(A) = \Sigma_k B(p_k/(p_k+n_k)).(p_k+n_k)/(p+n)$
  - **informační zisk** atributu

    $Gain(A) = B(p/(p+n)) - Remainder(A)$
  - $Gain(Patrons) \approx 0.541$ $Gain(Type) = 0$

Přeučení

- Někdy učící algoritmus vydá velký rozhodovací strom.
- **Příklad**: Chceme udělat rozhodovací strom, který určí, zda na kostce padne 6 podle atributů jako je hmotnost a barva kostky, držení palců apod.
  - dostaneme velký strom podle nalezených vzorů ve vstupních datech
  - rozumný strom by měl mít jeden uzel s odpovědí NE
- V takových případech hovoříme o **přeučení** (overfitting).
  - nastává obecně nejen u náhodných dat
  - větší možnost přeučení je u větších prostorů hypotéz a u větších počtů atributů
  - menší šance přeučení je u zvětšujícího se počtu tréninkových příkladů

Prořezání stromu



- Přeučení můžeme zmírnit **prořezáním** nalezeného rozhodovacího stromu.
  - vezmeme poslední rozhodovací uzel na nějaké cestě
  - pokud je atribut irelevantní (detekuje pouze šum v datech), uzel vyřadíme
- Jak poznáme **irelevantní atribut**?
  - **test významnosti** (χ^2 test)
    - uvažujme data bez vzoru (**nulová hypotéza**)
    - vypočteme odchylku aktuálních dat od nulové hypotézy

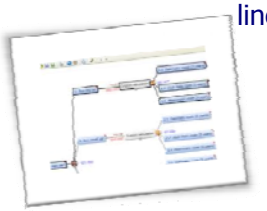
$$p`_k = p \cdot (p_k+n_k)/(p+n) \qquad n`_k = n \cdot (p_k+n_k)/(p+n)$$
$$\Delta = \Sigma_k (p_k-p`_k)^2/p`_k + (n_k-n`_k)^2/n`_k$$

    - použitím χ^2 tabulky najdeme hodnotu Δ zajišťující pro daný počet atributů (stupňů volnosti) přijetí či odmítnutí nulové hypotézy (přijetí znamená, že atribut je irelevantní)
- Můžeme χ^2 oříznutí použít už při konstrukci rozhodovacího stromu (**včasné zastavení**)?
  - raději ne, protože oříznutí se týká jednoho atributu, ale kombinace hodnot víc atributů může být informativní (například u stromu pro popis funkce XOR)

Rozhodovací stromy

praktické vlastnosti

- Pro reálné problémy potřebují rozhodovací stromy některá rozšíření resp. je potřeba vyřešit některé problémy:
  - **chybějící data** (mohou chybět hodnoty některých atributů)
    - Jak taková data klasifikovat? Jak upravit formuli pro výpočet informačního zisku?
    - vezmeme četnost výskytu hodnot a na jejím základě chybějící hodnoty doplníme
  - **mnoha-hodnotové atributy** (každý příklad může mít jinou hodnotu atributu)
    - informační zisk nemusí správně popisovat důležitost takových atributů
    - nemusíme větvení dělat podle všech hodnot, ale binárně podle jedné hodnoty
  - **spojité nebo číselné hodnoty vstupních atributů**
    - místo nekonečně mnoha větví se hodnoty atributů rozdělí podle vybraných bodů (příklady se setřídí podle hodnot atributu a za body dělení se berou hodnoty, kde se mění klasifikace)
    - výpočtově nejnáročnější část učení
  - **spojité hodnoty výstupního atributu**
    - bude potřeba **regresní strom**, tj. rozhodovací strom, který má v listech lineární funkci počítající na základě některých vstupních atributů výstupní hodnotu
    - učící se algoritmus musí rozhodnout, kdy ukončit dělení stromu a aplikovat lineární regresi



- Rozhodovací stromy mají prakticky velmi důležitou vlastnost – **je vidět, jak rozhodnutí bylo uděláno** (což třeba neplatí u neuronových sítí).

Regrese

- Jak naložíme se **spojitými proměnnými**?
- Prostorem hypotéz budou **lineární funkce**
  - začneme s lineárními funkcemi jedné proměnné (univariate linear regression)
  - zobecníme na lineární funkce více proměnných (multivariate linear regression)
  - nakonec ukážeme lineární klasifikátory (použití prahové funkce)

Lineární regrese

- Hledáme funkci ve tvaru $y = w_1.x + w_0$
- Označme $h_{\mathbf{w}}(x) = w_1.x + w_0$, kde $\mathbf{w} = [w_0,w_1]$
- Hledáme **hypotézu** $h_{\mathbf{w}}$, která nejlépe odpovídá datům (hledáme váhy w_1 a w_0).
- Jak měříme **chybu** vzhledem k datům?
  - Nejčastěji se používá kvadratická chyba

$$Loss(h_{\mathbf{w}}) = \Sigma_j (y_j - h_{\mathbf{w}}(x_j))^2 = \Sigma_j (y_j - (w_1.x_j + w_0))^2$$

- hledáme tedy $\mathbf{w}^* = \text{argmin}_{\mathbf{w}} Loss(h_{\mathbf{w}})$
  - lze najít vyřešením rovnic

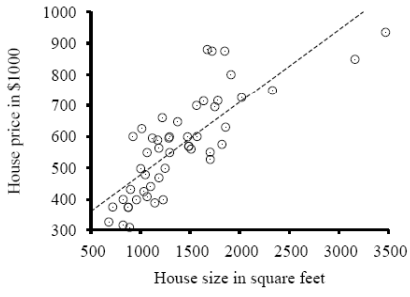
$$\partial/\partial_{w_0} \Sigma_j (y_j - (w_1.x_j + w_0))^2 = 0$$

$$\partial/\partial_{w_1} \Sigma_j (y_j - (w_1.x_j + w_0))^2 = 0$$

  - jednoznačné řešení

$$w_1 = (N \Sigma_j x_j y_j - \Sigma_j x_j \Sigma_j y_j) / (N \Sigma_j x_j^2 - (\Sigma_j x_j)^2)$$

$$w_0 = (\Sigma_j y_j - w_1. \Sigma_j x_j) / N$$



Nelineární regrese

- Pokud je prostor hypotéz definován jiným typem funkce, nemusí mít rovnice ∂/∂_{w_i} Loss($h_{\mathbf{w}}$) = 0 analytické řešení.
- Můžeme použít **metodu poklesu dle gradientu**
  - začneme s libovolným nastavením vah
  - upravíme váhy podle gradientu (největší pokles)

$$w_i \leftarrow w_i - \alpha \, \partial/\partial_{w_i} \text{Loss}(h_{\mathbf{w}}),$$

kde α je tzv. **learning rate** (může být konstantní nebo v průběhu učení klesat)

  - opakujeme do konvergence
- Pro lineární regresi dostaneme:

$$w_0 \leftarrow w_0 + \alpha \, \Sigma_j \, (y_j - h_{\mathbf{w}}(x_j))$$

$$w_1 \leftarrow w_1 + \alpha \, \Sigma_j \, (y_j - h_{\mathbf{w}}(x_j)).x_j$$

Regrese s více atributy

- Hypotéza je tvaru $h_{\mathbf{w}}(x) = w_0 + \Sigma_i \, w_i x_i$
- Příklady rozšíříme o nultý atribut s hodnotou 1

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

- Problém můžeme řešit analyticky, tj. řešením rovnic ∂/∂_{w_i} Loss($h_{\mathbf{w}}$) = 0

$$\mathbf{w}^* = (X^T X)^{-1} X^T y$$

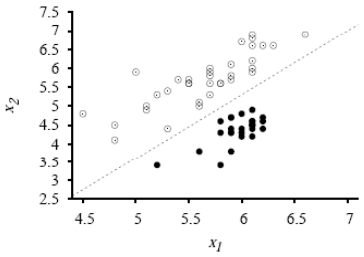
kde X je matice vstupních atributů tréninkových příkladů (řádek = příklad)

- Nebo použijeme metody poklesu dle gradientu

$$w_i \leftarrow w_i + \alpha \, \Sigma_j \, (y_j - h_{\mathbf{w}}(\mathbf{x}_j)).x_{j,i}$$

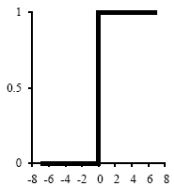
Lineární klasifikátor

- Lineární funkci můžeme také použít pro klasifikaci – pro oddělení dvou množin příkladů.
- **lineární separátor**
  - hledáme hypotézu h_w tž.
    - $h_w(\mathbf{x}) = 1$ pokud $\mathbf{w}.\mathbf{x} \geq 0$, jinak 0
  - alternativně můžeme funkci h definovat pomocí prahové funkce
    - $h_w(\mathbf{x})$ = Threshold($\mathbf{w}.\mathbf{x}$), kde Threshold(z) = 1, pro $z \geq 0$, jinak 0



- **perceptronové učící pravidlo**

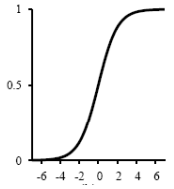
  $w_i \leftarrow w_i + \alpha\ (y - h_w(\mathbf{x})).x_i$
  - pokud je příklad klasifikován správně, váha se nemění
  - pokud $h_w(\mathbf{x}) \neq y$, potom se váha zvětší/zmenší dle x_i



- klasifikátor můžeme „změkčit" použitím **logistické prahové funkce**

  Threshold(z) = $1 / (1+e^{-z})$

  $w_i \leftarrow w_i + \alpha\ (y - h_w(\mathbf{x})).\ h_w(\mathbf{x}).(1-h_w(\mathbf{x})).x_i$
  - často používané v reálných aplikacích

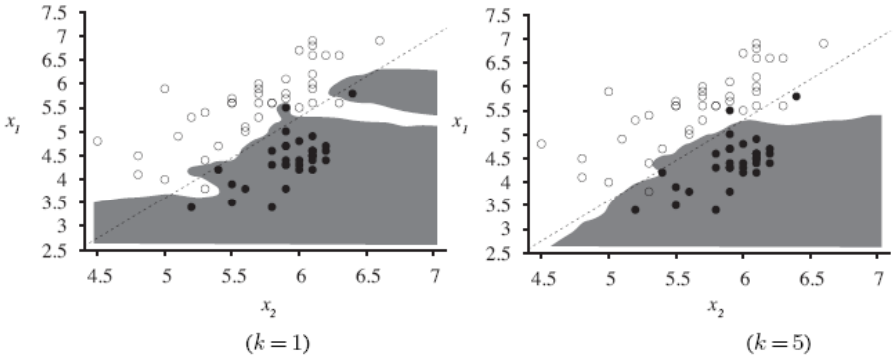


Neparametrické modely

- Když se lineární regrese naučí hypotézu, můžeme zahodit tréninková data.
- Hovoříme o tzv. **parametrických modelech** – data sumarizujeme do podoby omezeného předem daného počtu parametrů (vah).
- Když ale máme velké množství tréninkových příkladů, není lepší aby „mluvily sami za sebe" místo převodu do malého vektoru?
- Odpovědí jsou **neparametrické modely**, které nejsou charakterizovány vektorem parametrů, ale „pamatují" si tréninkové příklady.
- Pro nový příklad **x** vyhledáme v paměti stejný příklad a pokud tam je, vrátíme odpovídající y.
- Ale co když v paměti příklad se stejnými atributy není?

Nejbližší sousedé

- Jak najít odpověď pro příklad, který není v tabulce příkladů?
- Najdeme **k nebližších** tréninkových příkladů a odpověď složíme z jejich výstupů.
  - v případě klasifikace vybereme mezi nalezenými příklady třídy s největším výskytem (proto se k volí liché)



Nejbližší sousedé

vzdálenost

- Když hovoříme o nejbližších sousedech, jak definujeme vzdálenost?
- Typicky se používá tzv. **Minkowského vzdálenost**

$$L^p(\mathbf{x}_j,\mathbf{x}_q) = (\Sigma_i\ |x_{j,i} - x_{q,i}|^p)^{1/p}$$

  - p = 1: **Manhattanská vzdálenost**
  - p = 2: **Euclidovská** vzdálenost
  - Booleovské hodnoty: **Hammingova vzdálenost** (počet odlišných hodnot)
- Pozor na měřítko!
  - zpravidla se hodnoty **normalizují**
  - místo $x_{j,i}$ bereme $(x_{j,i} - \mu_i)/\sigma_i$, kde μ_i je průměr a σ_i je standardní odchylka

Neparametrické modely

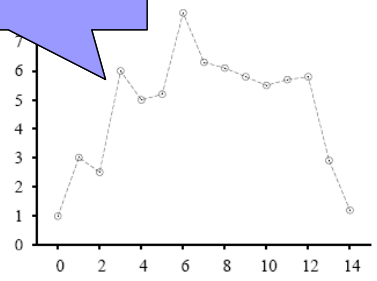
poznámky ke klasifikaci

- **Problém dimenzionality**
  - metoda nejbližších sousedů funguje dobře pro velké počty příkladů s malým počtem atributů (malou dimenzí)
  - v mnoho-dimenzionálním prostoru jsou nejbližší sousedé hodně daleko
- **Hledání sousedů**
  Jak vlastně najdeme k nejbližších sousedů?
  - **tabulka**: nalezení prvku trvá O(N)
  - **binární vyvážený strom**: nalezení prvku trvá O(log N), ale nejbližší sousedé mohou být ve vedlejších větvích
    - dobré, pokud je příkladů exponenciálně více než dimenzí
  - **hašovací tabulky**: nalezení prvku trvá O(1)
    - je potřeba hašovací funkce citlivá na vzdálenost (blízké prvky jsou ve stejném koši)
    - sousedy nelze hledat přesně, ale aproximačně (například použijeme více hašovacích tabulek, v každé najdeme prvky odpovídají příkladu a mezi nimi vybereme k nejbližších)

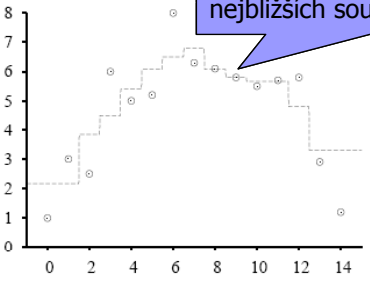
Neparametrická regrese

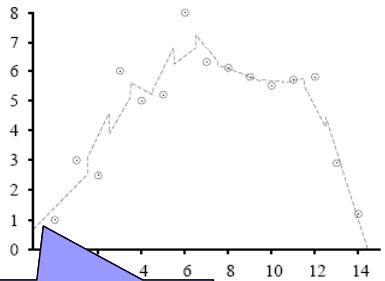
Podobně jako klasifikační úlohy můžeme řešit **regresní úlohu**.

Spojování bodů
lineární regrese dvou bodů z okolí

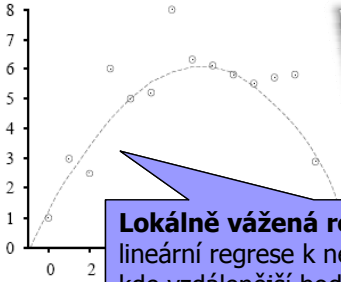


Průměr sousedů
vezmeme průměrný výstup k (3) nejbližších sousedů (není spojité)





Lineární regrese
uděláme lineární regresi k (3) nejbližších sousedů (není spojité)



Lokálně vážená regrese
lineární regrese k nejbližších sousedů, kde vzdálenější body mají menší váhu
váha určena **kernel funkcí** K
$\mathbf{w}^* = \text{argmin}_w \Sigma_j K(\text{Dist}(x_q,x_j))(y_j - \mathbf{w}.\mathbf{x}_j)^2$

Support Vector Machines

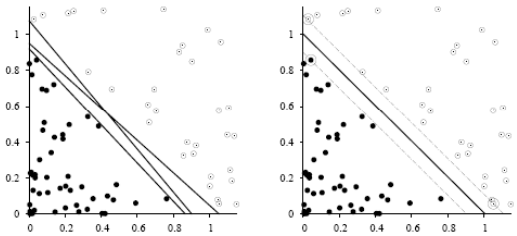
- SVM je dnes asi **nejrozšířenější metoda učení s učitelem**.
- Metoda je založená na třech základních vlastnostech:
  - hledání **lineárního separátoru** s maximálním okrajem (vzdáleností od příkladů)
  - pokud nelze příklady lineárně separovat, převedeme je **kernel funkcí** do jiného prostoru, kde separovat jdou
  - použijeme **neparametrický přístup** pro reprezentaci hypotézy (ale s málo příklady)

SVM separace

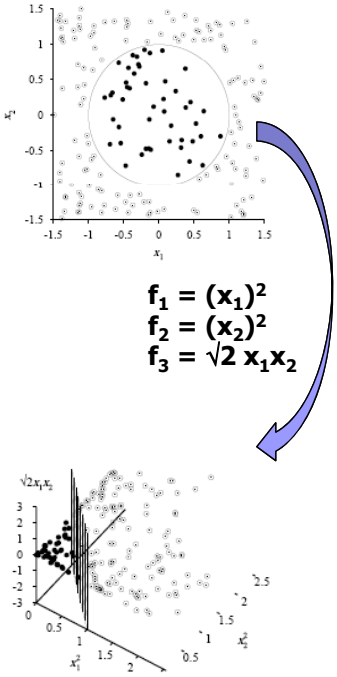
- Při hledání lineárního separátoru nejsou všechny tréninkové příklady stejně důležité!
- Ty, které se nacházejí na rozhraní mezi třídami, jsou významnější.
- Budeme hledat separátor, který má největší okraj (vzdálenost od tréninkových příkladů)
  - lze řešit přes **duální reprezentaci**

    $\text{argmax}_\alpha \sum_j \alpha_j - ½ \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j.\mathbf{x}_k)$, kde $\alpha_j \geq 0$, $\sum_j \alpha_j y_j = 0$
  - řešíme technikami kvadratického programování
  - všimněme si, že tréninkové příklady jsou při výpočtu brány po dvojicích
- Hypotéza je vyjádřena v duální reprezentaci

  $h(\mathbf{x}) = \text{sign}(\sum_j \alpha_j y_j (\mathbf{x}.\mathbf{x}_j) - b)$
- nebo pro původní reprezentaci $\mathbf{w} = \sum_j \alpha_j . \mathbf{x}_j$
- Důležitá vlastnost
  - hypotéza je sice reprezentována neparametricky tréninkovými příklady, ale pouze některé váhy jsou nenulové – váhy příkladů, které jsou nejblíže separátoru – tzv. **support vectors**
  - SVM tak spojuje výhody neparametrických modelů s parametrickými (držíme jen několik málo příkladů, kde $\alpha_j \neq 0$)



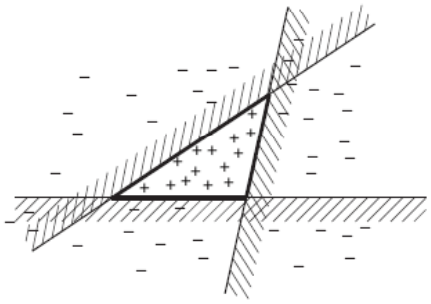
Kernel funkce



- Co když ale nejsou tréninkové příklady lineárně separovatelné?
- Můžeme vstupní vektor mapovat funkcí F do jiného prostoru, kde příklady budou separovatelné.
- Při hledání separace jednoduše místo x_j použijeme $F(x_j)$.
- Často nemusí F aplikovat na samostatné příklady, ale můžeme aplikaci provést rovnou po dvojicích.
  - $F(x_j)\ F(x_k) = (x_j.x_k)^2$
  - hovoříme o tzv. **kernel funkci** $K(x_j,x_k)$
  - používá se například polynomiální kernel $K(x_j,x_k) = (1+x_jx_k)^d$
    - dává prostor jehož dimenze je exponenciální v d

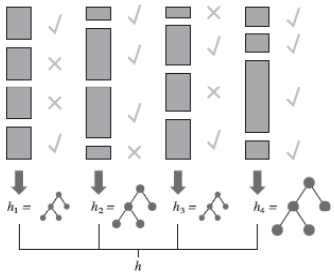
Skládání hypotéz

- Zatím jsem vždy hledali **jedinou hypotézu**.
- Co kdybychom hledali **více hypotéz** a jejich výsledky potom skládali?
  - hypotézy „hlasují" o výsledné klasifikaci – většina vyhrává – což **zvyšuje šanci na správnou klasifikaci**
  - přirozeně se tak také **zvětší prostor hypotéz**
    - můžeme například lineárním klasifikátorem klasifikovat lineárně neseparovatelné příklady



Boosting

- Metoda skládání hypotéz založená na vážení tréninkových příkladů a opakovaném hledání hypotézy:
  - na začátku dáme všem tréninkovým příkladů **stejnou váhu** (např. 1)
  - zvolenou metodou **najdeme** nějakou **hypotézu**
  - u příkladů, které jsou špatně klasifikovány **zvýšíme váhu**, o dobře klasifikovaných ji snížíme
  - **opakujeme** hledání hypotézy
  - získané hypotézy přispívají do výsledného klasifikátoru s vahou danou správnou klasifikací tréninkových příkladů
  - dokáže **zlepšit** i **slabé metody** učení (stačí když je základní metoda „o trochu" lepší než náhodné hádání výsledku)



AdaBoost

function ADABOOST($examples$, L, K) **returns** a weighted-majority hypothesis
  inputs: $examples$, set of N labeled examples $(x_1, y_1), \ldots, (x_N, y_N)$
    L, a learning algorithm
    K, the number of hypotheses in the ensemble
  local variables: $\mathbf{w}$, a vector of N example weights, initially $1/N$
    $\mathbf{h}$, a vector of K hypotheses
    $\mathbf{z}$, a vector of K hypothesis weights

  for $k = 1$ **to** K **do**
    $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
    $error \leftarrow 0$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
    $\mathbf{w} \leftarrow$ NORMALIZE($\mathbf{w}$)
    $\mathbf{z}[k] \leftarrow \log(1 - error)/error$
  return WEIGHTED-MAJORITY($\mathbf{h}, \mathbf{z}$)