



# **Kombinování klasifikátorů**

## ***Ensamble based systems***

# Rozhodování z více hledisek



- ❖ V běžném životě se často snažíme získat názor více expertů, než přijmeme závažné rozhodnutí:
  - ◆ Před operací se radíme s více lékaři
  - ◆ Před koupí nového vozu si přečteme recenze na několik různých produktů téže třídy
  - ◆ Při hledání místa nás zajímají reference na budoucího zaměstnavatele
- ❖ Proč nepoužít stejnou strategii i při automatizaci rozhodování?
- ❖ Tento přístup má ve strojovém učení mnoho různých jmen:
  - ◆ Multiple classifier systems,
  - ◆ committee of classifiers,
  - ◆ mixture of experts,
  - ◆ ensemble based systems

# Klasifikátor jako kombinace více klasifikátorů



Kombinace klasifikátorů (konzilium) si v řadě aplikací vede velmi úspěšně ve srovnání s jednodruhovými systémy

- ❖ Otázky spojené s návrhem kombinovaných klasifikátorů:
  - (i) jak volit individuální základní (*base*) klasifikátory ?
  - (ii) jak kombinovat jejich výsledky?

Příklady současných postupů

- ◆ **Bagging, boosting, AdaBoost**, postupná (stacked) generalizace a hierarchická směs expertů
- ◆ Běžně používaná **pravidla pro kombinaci výsledků**
- ◆ Algebraická kombinace, hlasování, *behavior knowledge space* & rozhodovací vzory (*decision templates*)

# Proč kombinovat klasifikátory ?



## ❖ **Statistické důvody**

- ◆ Množina různých klasifikátorů s podobným výkonem při trénování může mít rozdílné generalizační schopnosti
- ◆ Kombinací výsledků více klasifikátorů zmenšujeme riziko toho, že si zvolíme opravdu špatné řešení

## ❖ **Rozsáhlé objemy dat**

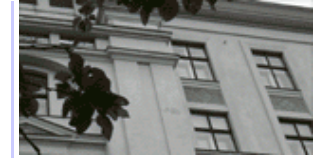
- ◆ Jeden SW systém nemusí pojmout všechna zpracovávaná data. Pak je vhodné použít současně více klasifikátorů na různé výběry z dat a jejich výsledky kombinovat.

## ❖ **Malé objemy dat**

- ◆ Kombinace je výhodná i v tomto případě: resampling techniques

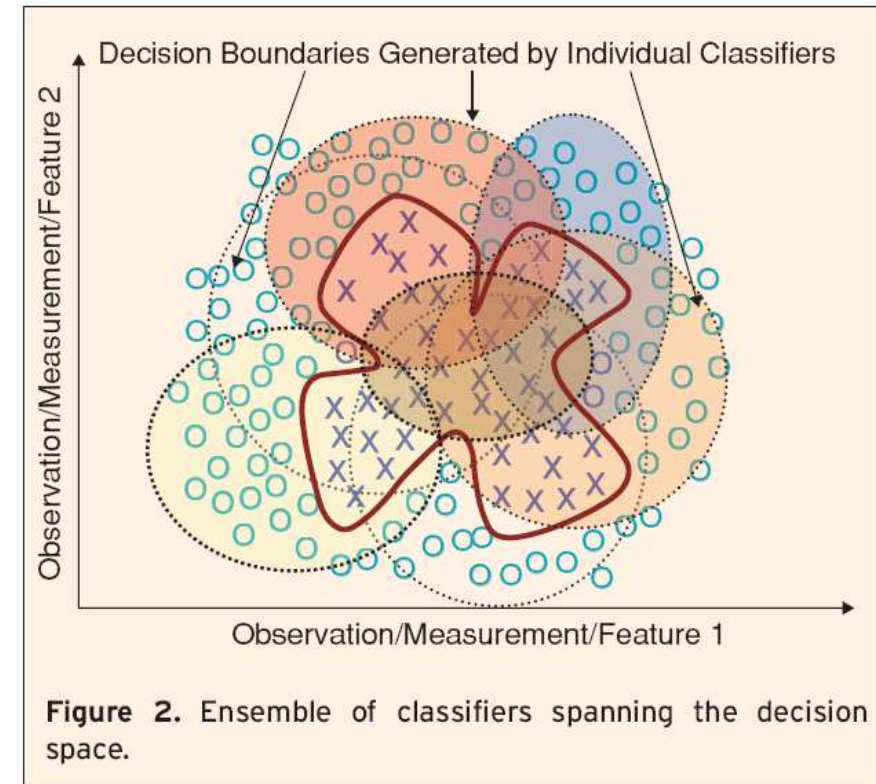
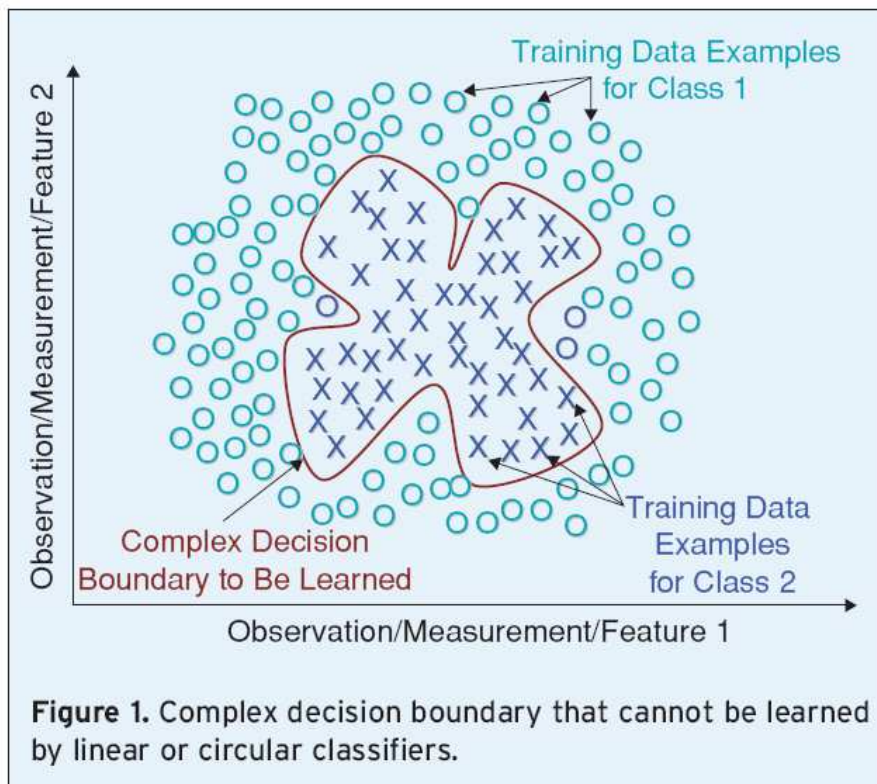
## ❖ **Nejednotný formát dat – viz data fusion**

# Proč kombinovat klasifikátory ?



## Rozděluj a panuj!

- ◆ Rozděľ data do menších lépe popsatelných skupin.
- ◆ Nechej každý klasifikátor zvládnout jen jednu z těchto skupin



# Kdy ještě kombinovat klasifikátory ?



## ❖ Sdružování dat (data fusion)

- ◆ Máme-li několik souborů dat z více zdrojů, v nichž jsou použity **různé typy atributů** zcela odlišného charakteru (heterogeneous features), může být naučení jediného klasifikátoru velmi obtížné (např. pro pacienta máme MRI data, EEG záznam, laboratorní testy,..)
- ◆ Problém s **chybějícími údaji!**
- ◆ Řada aplikací úspěšně využívá kombinací klasifikátorů pro rozhodování nad různorodými daty - data fusion applications

# Něco z historie



- ❖ První návrh postupu využívajícího kombinaci klasifikátorů: Dasarathy and Sheela (1979)
- ❖ Schapire (1990) dokázal, že velmi dobrý (*strong*) klasifikátor může vzniknout kombinací více průměrných (*weak*) klasifikátorů pomocí „boosting“; předchůdce algoritmu AdaBoost
- ❖ **2 přístupy ke vzniku různorodých klasifikátorů :**
  - ♦ **Výběr klasifikátoru** Každý klasifikátor je natrénován tak, aby se stal expertem v nějaké části definičního oboru uvažovaných příkladů; 1 či více lokálních expertů může být vyzváno, aby navrhli řešení (rozhodnutí) pro „svou část“. Jak vybírat vhodné části def.oboru?
  - ♦ **Fúze klasifikátorů** (*classifier fusion*). Všechny klasifikátory jsou trénovány nad celým definičním oborem; fúze pak znamená sdružení jednotlivých slabších klasifikátorů tak, aby vznikl jeden výrazně lepší.
- ❖ **Jak kombinovat výsledky?**

# Různorodost použitých klasifikátorů



❖ **Cíl:** vytvořte mnoho klasifikátorů vhodných pro kombinování.

**Intuice:** Pokud každý klasifikátor dělá poněkud jiný typ chyb, pak jejich kombinace může být výhodná!

Potřebujeme tedy klasifikátory takové, že

- ♦ dávají relativně dobré výsledky,
  - ♦ mají vzájemně dost různé rozhodovací hranice (*dec. boundaries*)
- Taková množina klasifikátorů se nazývá různorodá (*diverse*)

❖ **Jak zajistit vznik různorodé množiny**

**klasifikátorů?** Individuální klasifikátory jsou trénovány nad různými trénovacími množinami dat, které jsou získány např. pomocí technik jako

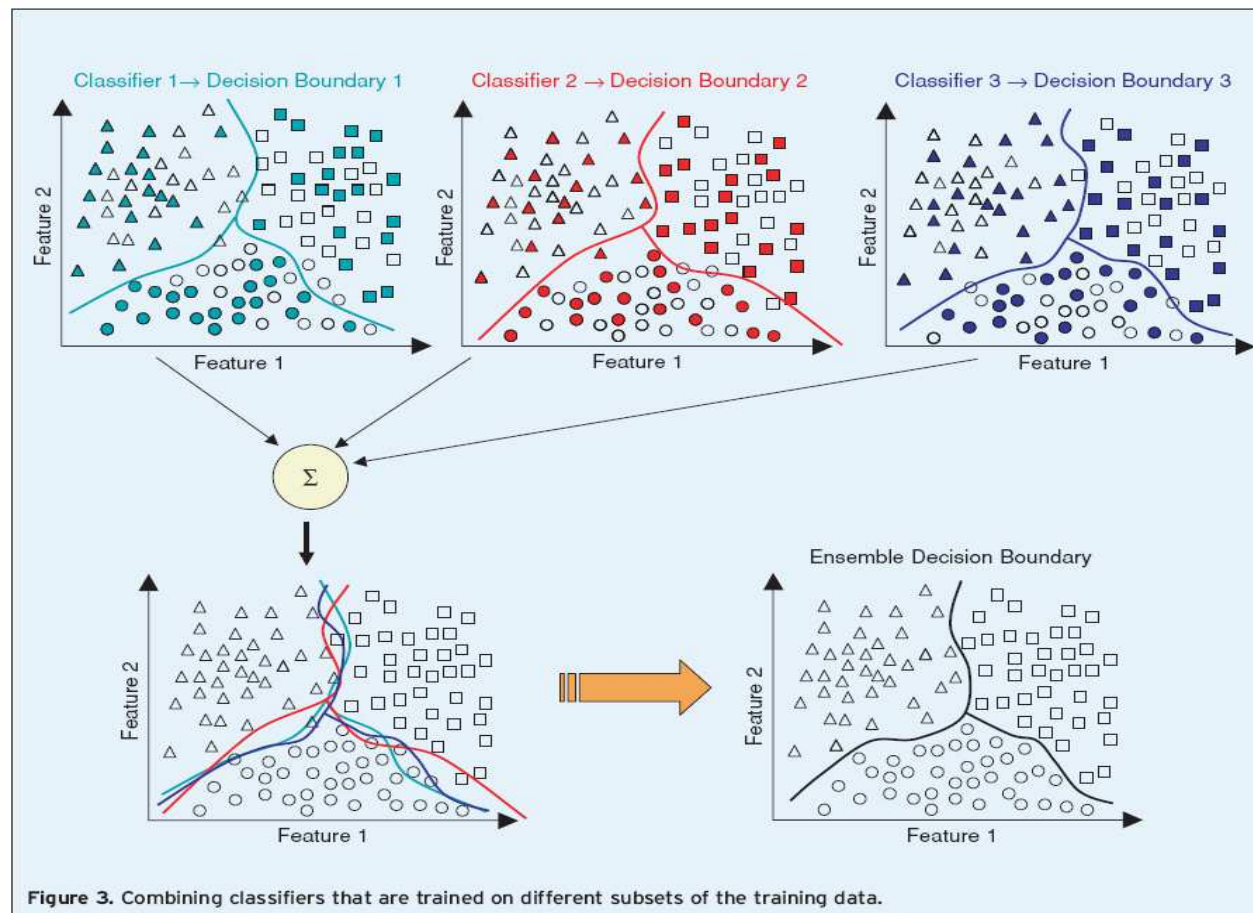
- ❖ **Násobné dělení**
- ❖ **Vzorkovací (*resampling*) techniky:** bootstrapping nebo bagging – trénovací podskupiny jsou vybírány náhodně (s navracením) z výchozí dat



# Vzorkování s navrácením



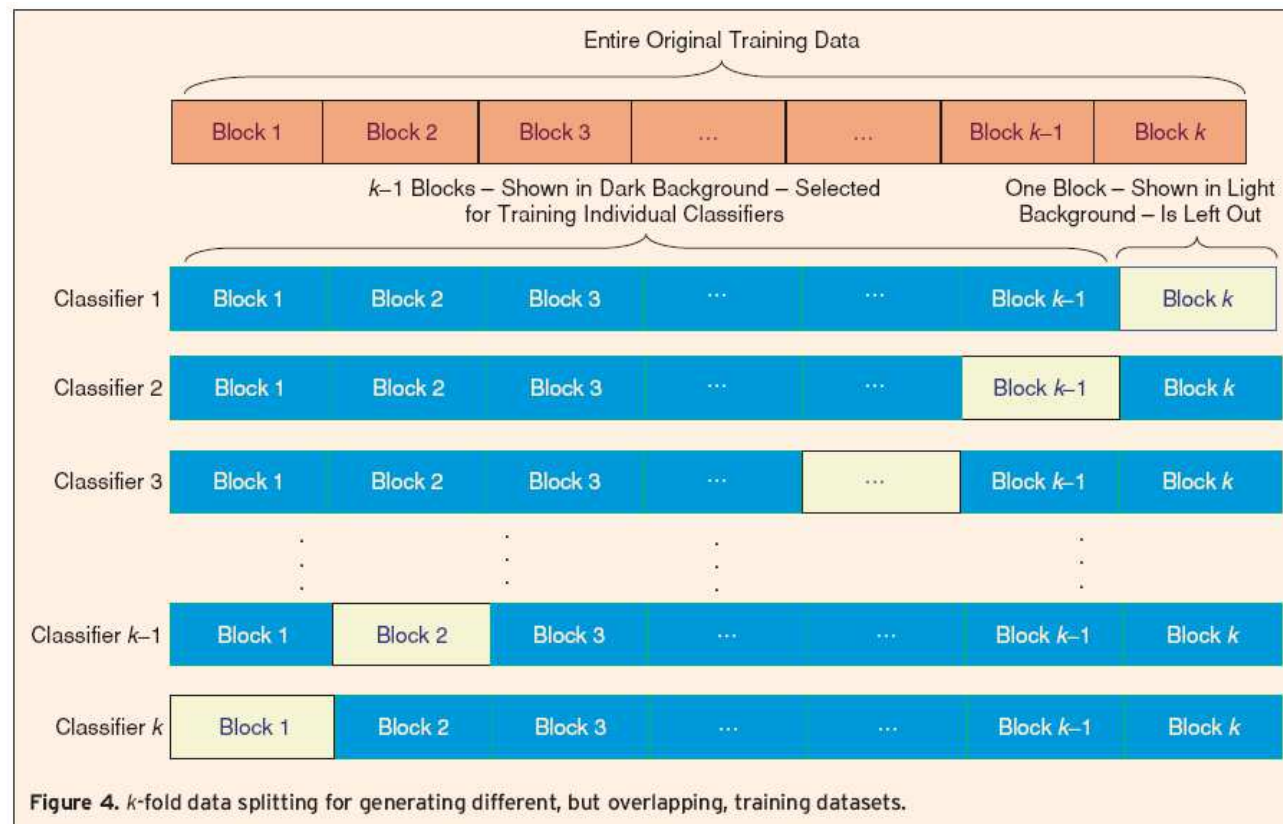
Náhodně vybrané & překrývající se trénovací skupiny jsou použity pro vznik 3 klasifikátorů, jejichž rozhodnutí se pak kombinuje v jediném, které je přesnější než každé z nich samostatně!



# Jackknife čili $k$ -násobné rozdělení dat



Celý výchozí soubor dat je rozdělen do  $k$  stejně velkých disjunktních bloků; každý klasifikátor je trénován na skupině dat vzniklé z původních vynecháním jednoho bloku



# † Další cesty k různorodosti



- ❖ Kombinace rozhodnutí **různých typů klasifikátorů** (např. vícevrstvý perceptron - MLP, rozhodovací strom, neuronová síť NN, SVM)
- ❖ Tentýž typ klasifikátoru lze použít s **různými výchozími hodnotami parametrů**
  - ◆ U MLP nebo NN je rozdíl dán výchozími vahami různých hran, volbou architektury, atd.
  - ◆ U rozhodovacích stromů můžeme měnit volbu diskretizace atributů, kritérium větvení, způsob prořezávání stromu, maximální hloubku stromu ...
  - ◆ Změny výchozích parametrů ovlivňují (ne)stabilitu výsledného klasifikátoru (lokální minimum)
- ❖ Různorodosti lze dosáhnout i tím, že pro učení nepoužijeme popis objektu všemi atributy, ale jen jejich náhodně vybranou částí:  
**metoda náhodného podprostoru** (*random subspace*)

# Bagging, zkratka za bootstrap aggregating



Jedna z nejstarších metod pro tvorbu souboru klasifikátorů

- ❖ Velmi intuitivní metoda jednoduchá na implementaci, s překvapivě dobrými výsledky:
  - ◆ **Velké množství** (řekněme 200) trénovacích podmnožin je náhodně vybráno – s náhradou – z výchozích trénovacích dat
  - ◆ Každá z těchto trénovacích podmnožin je použita pro vygenerování klasifikátoru (tentýž typ pro všechny podmnožiny)
  - ◆ Výsledné jednotlivé **klasifikátory se kombinují hlasováním**
- ❖ Bagging je zvláště **atraktivní pro malé soubory trénovacích dat**, protože relativně velká část dat se dostane do každé trénovací podmnožiny

# Pseudokód pro bagging



**Vstup:**  $E := \emptyset$

Správně klasifik. trénovací data  $S$  zařazená do tříd z množiny  $\Omega = \{\omega_1, \dots, \omega_C\}$ .

Alg. stroj. učení **ASU**. Celé číslo  $T$  rovné počtu iterací a  $F$  díl pro bootstrap. \*

**Dělej**  $t = 1, \dots, T$

- Vytvoř pomocí bootstrapping repliku  $S_t$  trén. dat  $S$  odpovídající dílu  $F$
- Aplikací **ASU** na  $S_t$  vytvoř hypotézu (klasifikátor)  $h_t$
- Přidej klasifikátor  $h_t$  do vytvářeného souboru klasifikátorů)  $E$

**Použití souboru  $E$  pro hlasování** o zařazení neklasifikované instance  $x$ :

- Necht'  $v_{t,j} = 1$  právě tehdy, když hypotéza  $h_t$  zařadí  $x$  do třídy  $j$  (jinak je 0)
- Vypočti celkový počet hlasů  $V_j = \sum_{t=1..T} v_{t,j}$  pro jednotlivé třídy  $j = 1, \dots, C$
- Zařad'  $x$  do třídy, jejíž počet hlasů je nejvyšší

\*  $F$  určuje, kolik prvků budeme brát z původní množiny mohutnosti  $N$  v %:  
při metodě **bootstrapping** vybíráme  $N * F / 100$  prvků s navrácením.

# Variace na téma „bagging“



**Náhodný les** vzniká kombinací klasifikátorů ve tvaru rozhodovacích stromů s odlišnou volbou trénovacích parametrů

- ◆ Tyto parametry mohou odpovídat změně trénovacích dat jako je tomu u bagging (*bootstrapped replicas of the training data*),
- ◆ Nebo také mohou být důsledkem volby jiné podmnožiny sledovaných atributů jako v případě metod s náhodným podprostorem (*random subspace methods*)
- ◆ ...

# Boosting (vylepšování)

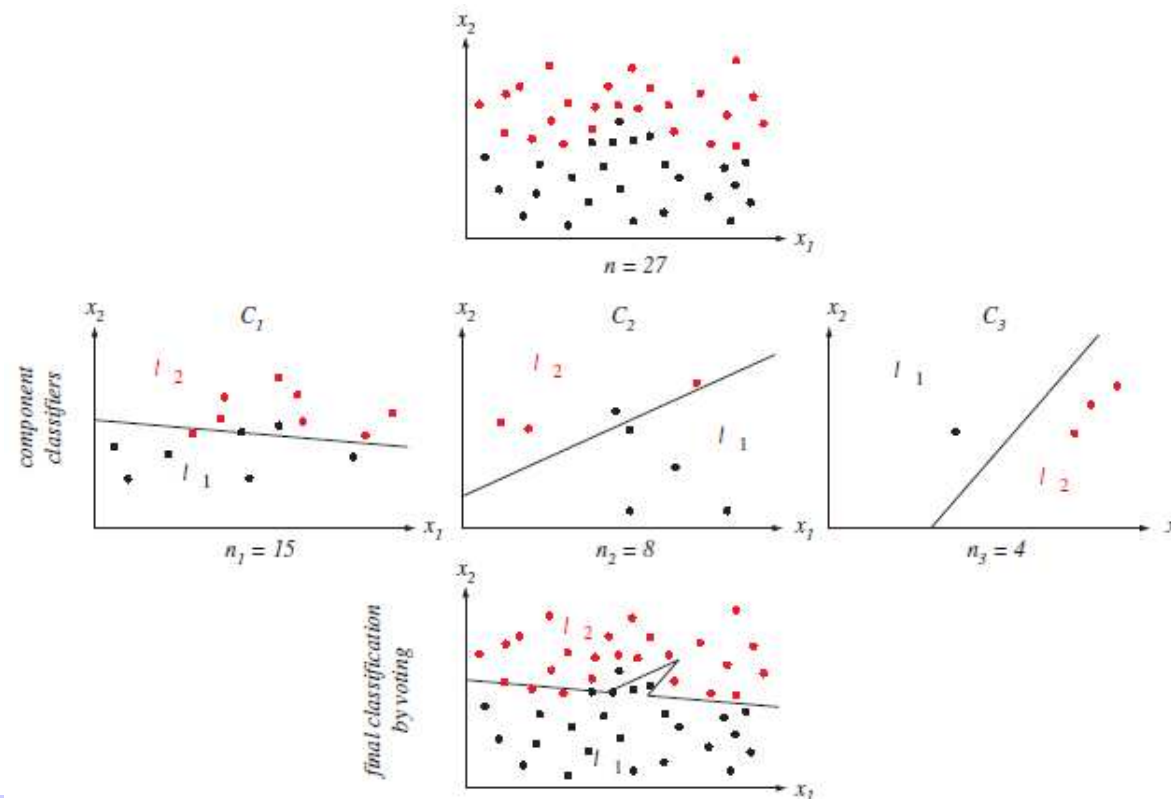


- ❖ Snaží se vylepšit výkon slabého klasifikátoru (*weak learner*) a tak udělat z něj udělat silný (*strong*) klasifikátor
- ❖ **Boosting** se opírá o soubor binárních klasifikátorů, které vznikají cíleným vzorkováním dat a jsou kombinovány pomocí hlasování
  - ◆ Postupné vzorkování je strategicky zaměřováno tak, aby každý nový vzorek obsahoval data nejpodstatnější pro další rozhodnutí
- ❖ Boosting vytváří **3 slabé** (dobré, ale ne skvělé) **klasifikátory**:
  1. První klasifikátor **C1** je natrénován na náhodně vybrané podmnožině **T1** výchozích trénovacích dat
  2. Trénovací množina **T2** pro tvorbu druhého klasifikátoru **C2** je vybrána jako **nejinformativnější množina** vzhledem k výsledkům klasifikátoru **C1**: polovinu dat v **T2** klasifikuje **C1** správně, druhou polovinu tvoří ty příklady, na kterých **C1** dělá chybu
  3. Třetí klasifikátor **C3** je natrénován na těch instancích trénovacích dat, na kterých se klasifikace **C1** & **C2** liší

# Boosting



- ❖ Uvažujme případ, kdy **ASU** (alg.stroj.učení) hledá **lineární hranici** postupem minimalizace čtverců odchylek
- ❖ Hlasující trojice má vždy lepší klasifikační přesnost než každý z klasifikátorů samostatně





# AdaBoost



- ❖ **AdaBoost** (1997) zobecňuje boosting. **AdaBoost.M1** pracuje dokonca s problémy, kde se vyskytuje více tříd
- ❖ AdaBoost generuje množinu hypotéz (klasifikátorů), které kombinuje **váženým hlasováním** o zařazení do tříd tak, jak je navrhuje jednotlivé klasifikátory
- ❖ Hypotézy vznikají natrénováním slabších klasifikátorů nad **vzorky dat, které iterativně mění distribuci trénovacích dat** tak, že postupně **zvýhodňují dosud špatně klasifikované instance**: Tedy postupně vznikající klasifikátory jsou trénovány na stále obtížněji klasifikovatelných instancích dat

# Pseudokod pro AdaBoost.M1



## Algoritmus AdaBoost.M1

**Vstup** : Mnozina  $N$  prikladu  $S = [(x_i, y_i)]_{i=1, \dots, N}$  klas. do trid  $y_i \in \Omega, \Omega = \{\omega_1, \dots, \omega_c\}$

Slaby algoritmus strojoveho uceni ASU ;

Parametr  $T$  specifikuj ici planovany pocet iteraci.

**Inicializace**  $D_1(i) = \frac{1}{N}, i = 1, \dots, N$

**Dalej pro**  $t = 1, 2, \dots, T$  :

1. Vyber tren ovaci podmnozinu  $S_t$ , vybranou podle distribuce  $D_t$ .

2. Pouzij ASU na  $S_t$  a vytvor ta k hypotezu (klasifikator)  $h_t$ .

3. Vypocti chybu vysl edne hypotezy na  $S$

$$h_t : \varepsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$$

Je - li  $\varepsilon_t > 1/2$ , **KONEC** .

4. Poloz  $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$

5. Uprav distribucni funkci tak, ze se snizi vaha spravne klasifikov anych prikladu

$$D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \begin{cases} \beta_t, & \text{pro } h_t(x_i) = y_i \\ 1, & \text{jinak} \end{cases}$$

kde  $Z_t$  je normalizac ni konstanta zvolena tak, aby  $\sum_i D_{t+1}(i) = 1$ , t.j. aby slo o pravedpodo bnostni rozlozeni.

**Zaverecne rozhodnuti - vazene hlasovani** : pro neklasifik ovanou instanci  $x$ .

1. Ziskej celkovy pocet hlasu pro jednotlivé tridy vaz eny "kvalitou" klasifikat oru

$$V_j = \sum_{i: h_t(x) = \omega_j} \log \frac{1}{\beta_t}, j = 1, \dots, C.$$

<#>

2. Instanci  $x$  prirad tridu s nejvyssi vahou!

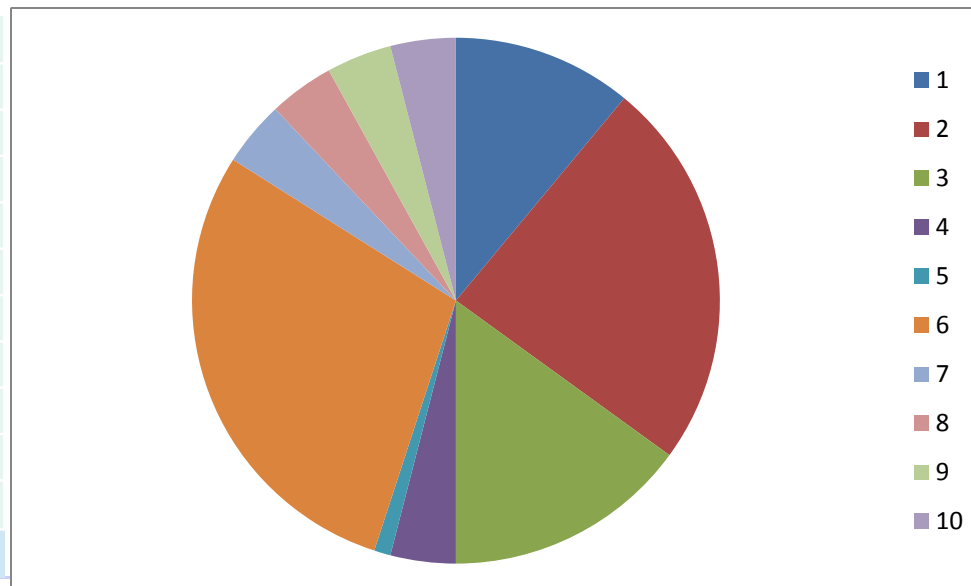


# Výběr podle distribuční funkce: Jak se používá princip ruletového kola?

Každému prvku  $x_i$  výchozí množiny je přidělena vlastní kruhová výseč o ploše úměrné hodnotě distribuční funkce pro tento prvek  $D(x_i)$ . Výsledek je podobný koláčovému grafu, viz obr. Dole.

Kolo se roztáčí  $N$  krát a při každém zastavení vybere 1 prvek (jedná se o výběr s navracením) – větší pravděpodobnost výběru mají prvky s vyšší hodnotou distribuční funkce.

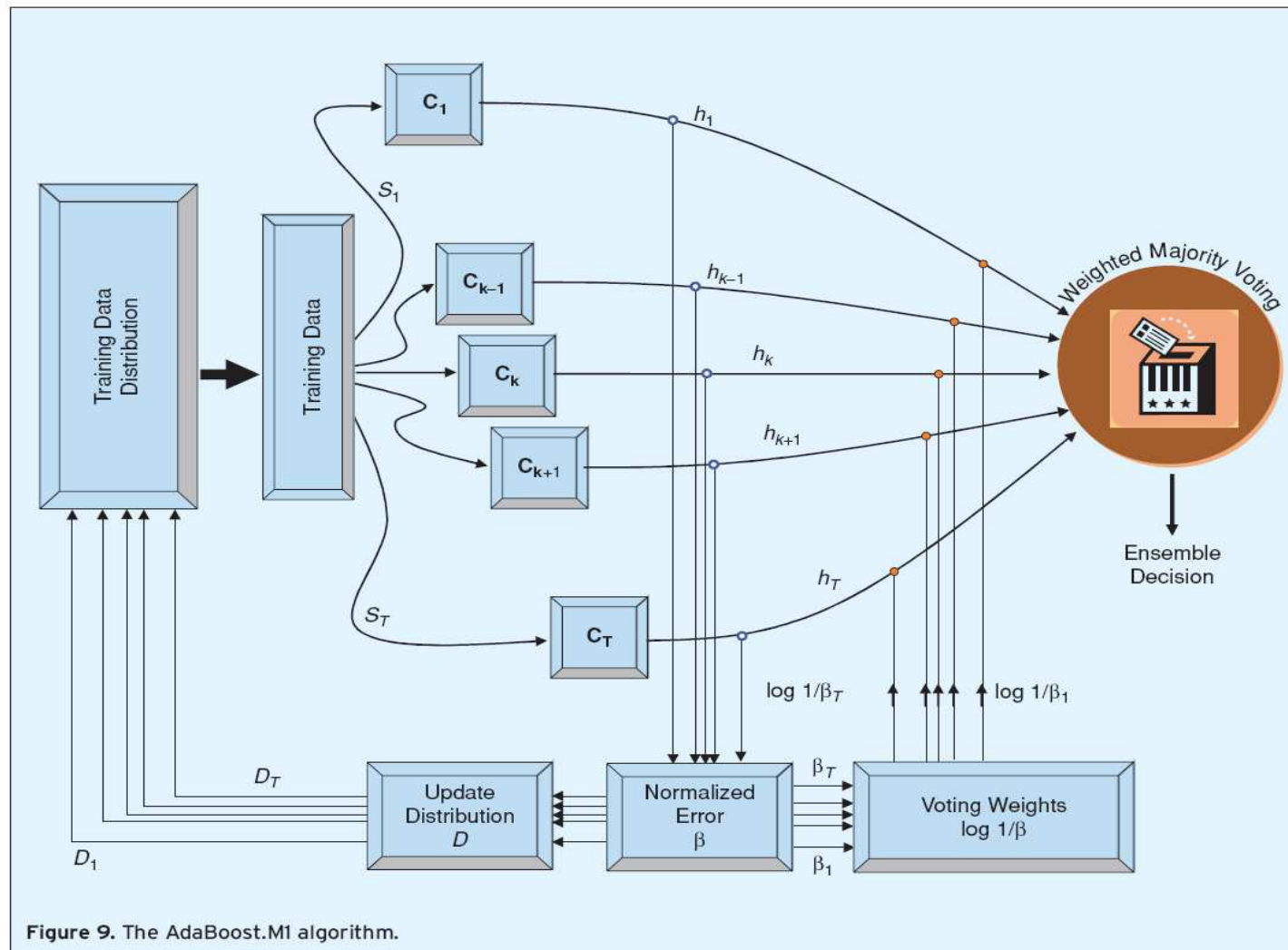
i	D(x <sub>i</sub> )
1	0,1
2	0,25
3	0,15
4	0,03
5	0,01
6	0,3
7	0,04
8	0,04
9	0,04
10	0,04
celkem	1,00



# AdaBoost.M1



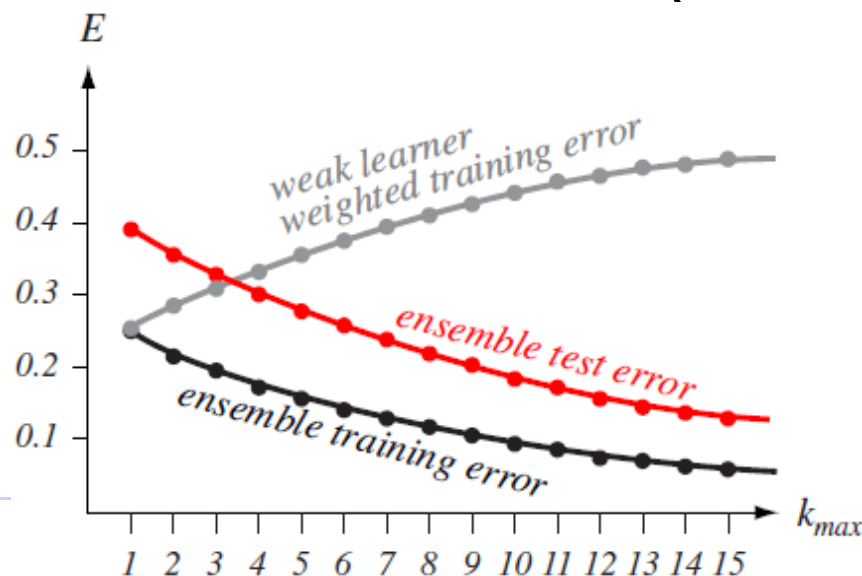
AdaBoost algoritmus **pracuje sériově**: klasifikátor ( $C_{K-1}$ ) vzniká dřív než klasifikátor  $C_K$



# AdaBoost



- ◆ AdaBoost se slabým klasifikátorem vede s rostoucím počtem kroků  $k_{max}$  k **exp.poklesu velikosti chyby** výsledného klasifikátoru.
- ◆ Naopak chyba sam.postupně vytvářených klasifikátorů roste (šedá čára), neboť tyto klasifikátory musí řešit stále těžší úlohy. Přesto jejich **kombinace dosahuje stále menší chyby na trénovacích datech** (černá čára), **pokud chyba jednotlivých klasifikátorů je menší než 0.5**.
- ◆ Vývoj **chyby pozorovaný na testovacích datech** má velmi často podobný trend jako na datech trénovacích (červená čára).



# Výkonnost pro AdaBoost



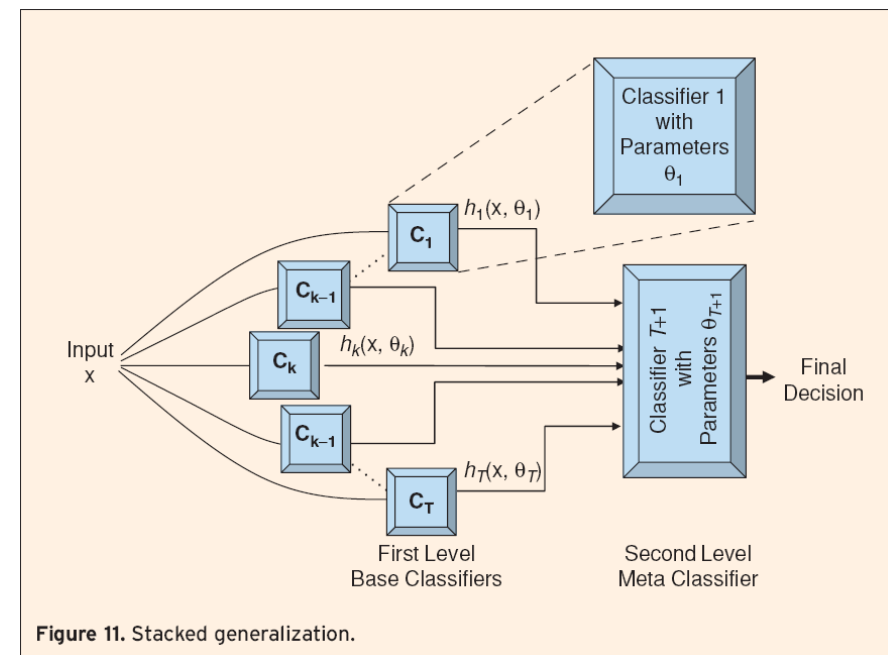
- ❖ Ve většině praktických příkladů, chyba klasifikace prostřednictvím souboru klasifikátorů rychle klesá během prvních několika iterací a konverguje k nějaké setrvalé hodnotě (ideálně k 0)
- ❖ **Empirické pozorování:** metoda AdaBoost nevede k „přeučení“ (*overfitting*); pokus o vysvětlení nabízí *margin theory*

# Stacked Generalization



se snaží rozpoznat spolehlivost jednotlivých vytvořených klasifikátorů (modelů) tak, že konstruuje **meta-klasifikátor**: Na **1. úrovni** je vytvořen soubor klasifikátorů, jejichž vektor klasifikací (doplněný správnou klas.) tvoří trénovací data pro **2. úroveň**: návrh meta-klasifikátoru, který z výsledků klasifikací souborem klasifikátorů navrhuje výslednou klasifikaci

- $C_1, \dots, C_T$  jsou natrénovány s využitím různých parametrů  $\theta_1$  až  $\theta_T$  a tak vzniknou hypotézy (klasifikátory)  $h_1$  až  $h_T$
- Výstupy těchto klasifikátorů doplněné o správnou klasifikaci představují trénovací data pro vznik meta-klasifikátoru  $C_{T+1}$



# Shrnutí



- ❖ Soubory klasifikátorů jsou prakticky velmi užitečné
- ❖ Pro úspěch je důležité zajistit rozdílnost použitých klasifikátorů
- ❖ Techniky pro vytvoření souboru: bagging, AdaBoost, směs expertů, využití dat bez uniformní reprezentace (data fusion),...
- ❖ Strategie pro kombinování výsledků klasifikátorů: algebraické postupy, hlasování, rozhodovací vzory.
- ❖ **Žádná technika ani strategie není vždy nejlepší !**

## **Další informace:**

Polikar R., "Ensemble Learning," Scholarpedia, 2008.

Berka P.: *Dobývání znalostí z databází*, Academia 2003

Rolach Lior: Ensemble-based classifiers, *AI Review*, Vol.33 (2010), pp.1-39

(<http://link.springer.com/article/10.1007%2Fs10462-009-9124-7>)