

Instrumente Inteligente
pentru
Bunăstare Socială
(Intelligent Tools for Social Good - ITSG)



Tema 3

Laura Dioșan

Conținut

- ❑ Instruire automata (Machine Learning - ML)
 - Problematică
 - Proiectarea unui sistem de învățare automată
 - Tipologie
 - ❑ Învățare supervizată
 - ❑ Învățare nesupervizată
 - ❑ Învățare cu întărire
 - ❑ Teoria învățării
- ❑ De citit:
 - S.J. Russell, P. Norvig – Artificial Intelligence - A Modern Approach → capitolul 18, 19, 20
 - www.google.com

Problematică

□ Date → Inteligență

■ Informal:

□ Intrări x → Program → Ieșiri Y

□ Similar:

- Reprezentarea relației dintre intrări și ieșiri
- Inferența ieșirilor din intrări
- Învățarea celui mai bun model care descrie datele

■ Formal:

□ Îmbunătățirea task-ului T

- stabilirea scopului (ceea ce trebuie învățat) - funcției obiectiv – și reprezentarea sa
- alegerea unui algoritm de învățare care să realizeze inferența (previziunea) scopului pe baza experienței

□ respectând o metrică de performanță P

- evaluarea performanțelor algoritmului ales

□ bazându-se pe experiența E

- alegerea bazei de experiență

Problematică

- Se cunosc sau nu ieșirile asociate unor date de antrenament?
 - Învățare supervizată – se cunosc
 - Învățare nesupervizată – nu se cunosc
 - Învățare semi-supervizată – se cunosc parțial

- Tipul de ieșiri
 - Valori discrete → Clasificare
 - Valori real → Regresie
 - Grupuri → Clusterizare

Proiectarea unui sistem de învățare automată

- Îmbunătățirea task-ului T
 - stabilirea scopului (ceea ce trebuie învățat) - funcției obiectiv – și reprezentarea sa
 - alegerea unui algoritm de învățare care să realizeze inferența (previziunea) scopului pe baza experienței
- respectând o metrică de performanță P
 - evaluarea performanțelor algoritmului ales
- bazându-se pe experiența E
 - alegerea bazei de experiență

Proiectare – Alegerea funcției obiectiv

- Care este funcția care trebuie învățată?
 - Ex. pt jocul de dame
 - o funcție care
 - alege următoarea mutare
 - evaluează o mutare
 - obiectivul fiind alegerea celei mai bune mutări

Proiectare – Reprezentarea funcției obiectiv

- Diferite reprezentări
 - tablou (tabel)
 - reguli simbolice
 - funcție numerică
 - funcții probabilistice
 - ex. jocul de dame
 - Combinație liniară a nr. de piese albe, nr. de piese negre, nr. de piese albe compromise la următoarea mutare, nr. de piese negre compromise la următoarea mutare

- Există un compromis între
 - expresivitatea reprezentării și
 - ușurința învățării

- Calculul funcției obiectiv
 - timp polinomial
 - timp non-polinomial

Proiectare – Alegerea unui model de învățare

- Metodologia (procesul) de lucru
 - folosind datele de antrenament
 - induce definirea unor ipoteze care
 - să se potrivească cu datele de antrenament și
 - să generalizeze cât mai bine datele ne-văzute (datele de test)

- Principiul de lucru
 - minimizarea unei erori (funcție de cost – *loss function*)

Proiectare – Învățare automată – tipologie

- **Învățare supervizată**
- Învățare nesupervizată
- Învățare cu întărire

Învățare supervizată

Scop:

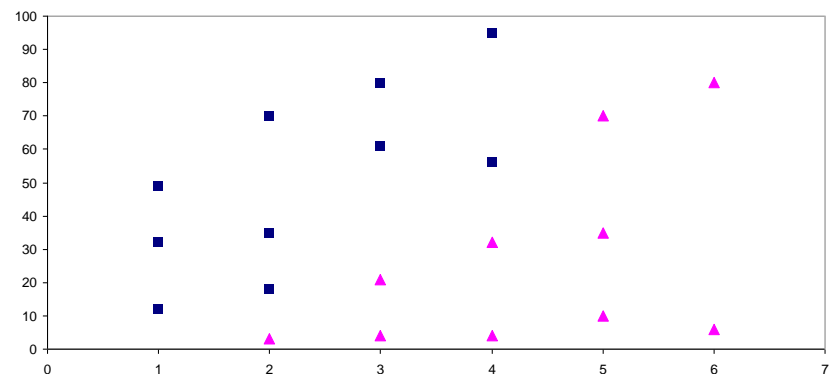
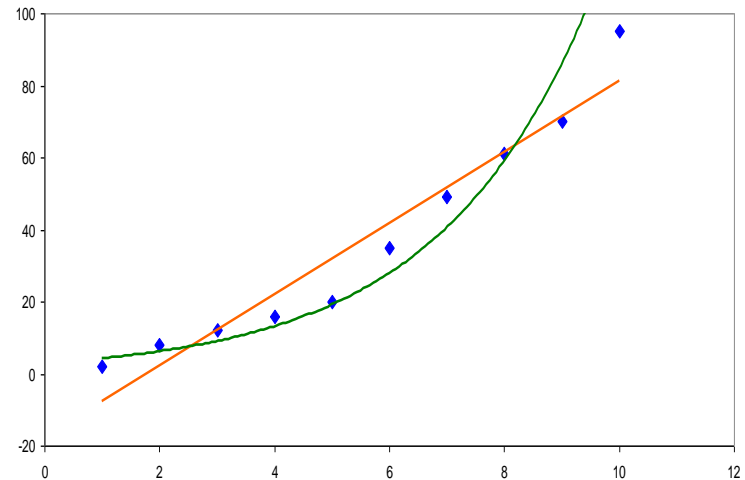
- Furnizarea unei ieșiri corecte pentru o nouă intrare

Tip de probleme

- regresie
 - Scop: predicția output-ului pentru un input nou
 - Output continuu (nr real)
 - Ex.: predicția prețurilor
- clasificare
 - Scop: clasificarea (etichetarea) unui nou input
 - Output discret (etichetă dintr-o mulțime predefinită)
 - Ex.: detectarea tumorilor maligne

Caracteristic

- BD experimentală adnotată (pt. învățare)



Învățare supervizată – definiere

Definire

□ Se dă

- un set de date (exemple, instanțe, cazuri)
 - date de antrenament – sub forma unor perechi ($attribute_data_i, ieșire_i$), unde
 - $i = 1, N$ ($N = \text{nr datelor de antrenament}$)
 - $attribute_data_i = (atr_{i1}, atr_{i2}, \dots, atr_{im})$, m – nr atributelor (caracteristicilor, proprietăților) unei date
 - $ieșire_i$
 - o categorie dintr-o mulțime dată (predefinită) cu k elemente (k – nr de clase) → problemă de clasificare
 - un număr real → problemă de regresie
 - date de test
 - sub forma ($attribute_data_i$), $i = 1, n$ ($n = \text{nr datelor de test}$).

□ Să se determine

- o funcție (necunoscută) – ipoteză – care realizează corespondența atribut – ieșire pe datele de antrenament
- ieșirea (clasa/valoarea) asociată unei date (noi) de test folosind funcția învățată pe datele de antrenament

Alte denumiri

- Clasificare (regresie), învățare inductivă

Învățare supervizată – exemple

- Recunoașterea scrisului de mână
- Recunoașterea imaginilor
- Previziunea vremii
- Detecția spam-urilor
- Recomandări de produse
- Traduceri automate

Învățare supervizată – proces

Procesul

- 2 pași:
 - Antrenarea
 - Învățarea, cu ajutorul unui algoritm, a modelului de clasificare
 - Testarea
 - Testarea modelului folosind date de test noi (*unseen data*)

Calitatea învățării

- o măsură de performanță a algoritmului → ex. acuratețea
 - $\text{Acc} = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$
- calculată în:
 - faza de antrenare
 - Ansamblul de date antrenament se împarte în
 - Date de învățare
 - Date de validare
 - Performanța se apreciază pe sub-ansamblul de validare
 - O singură dată
 - De mai multe ori → validare încrucișată (cross-validation)
 - faza de testare
- probleme
 - Învățare pe derost (*overfitting*) → performanță bună pe datele de antrenament, dar foarte slabă pe datele de test

Învățare supervizată – evaluare

Metode de evaluare

- Seturi disjuncte de antrenare și testare
 - pt. date numeroase
 - setul de antrenare
 - poate fi împărțit în
 - Date de învățare
 - Date de validare
 - Folosit pentru estimarea parametrilor modelului
 - Cei mai buni parametri obținuți pe validare vor fi folosiți pentru construcția modelului final

- Validare încrucișată cu mai multe (h) sub-seturi ale datelor (de antrenament)
 - separarea datelor de h ori în
 - $h-1$ sub-seturi pentru învățare
 - 1 sub-set pt validare
 - dimensiunea unui sub-set = dimensiunea setului / h
 - performanța este dată de media pe cele h rulări
 - $h = 5$ sau $h = 10$
 - pt date puține

- Leave-one-out cross-validation
 - similar validării încrucișate, dar $h = \text{nr de date}$ → un sub-set conține un singur exemplu
 - pt. date foarte puține

Învățare supervizată – evaluare

Măsuri de performanță

- Măsuri statistice
 - Funcții de scor (acuratețe, precizie, etc.)
 - Funcții de loss (măsoară calitatea modelului învățat)

- Eficiența
 - În construirea modelului
 - În testarea modelului

- Robustețea
 - Tratarea zgomotelor și a valorilor lipsă

- Scalabilitatea
 - Eficiența gestionării seturilor mari de date

- Interpretabilitatea
 - Modelului de clasificare

- Proprietatea modelului de a fi compact

- Scoruri

Învățare supervizată – evaluare

Măsuri de performanță

□ Măsuri statistice

■ Acuratețea

- Nr de exemple corect clasificate / nr total de exemple
- Opusul erorii
- Calculată pe
 - Setul de validare
 - Setul de test

□ Uneori

- Analiză de text
- Detectarea intrușilor într-o rețea
- Analize financiare

este importantă doar o singură clasă (clasă pozitivă) → restul claselor sunt negative

Învățare supervizată – evaluare

Măsuri de performanță

□ Măsuri statistice

■ Precizia și Rapelul

□ Precizia (P)

- nr. de exemple pozitive corect clasificate / nr. total de exemple clasificate ca pozitive
- probabilitatea ca un exemplu clasificat pozitiv să fie relevant
- $TP / (TP + FP)$

□ Rapelul (R)

- nr. de exemple pozitive corect clasificate / nr. total de exemple pozitive
- Probabilitatea ca un exemplu pozitiv să fie identificat corect de către clasificator
- $TP / (TP + FN)$

□ Matricea de confuzie

- Rezultate reale vs. rezultate calculate

■ Scorul F1

- Combină precizia și rapelul, facilitând compararea a 2 algoritmi
- Media armonică a preciziei și rapelului
- $2PR / (P + R)$

		Rezultate reale	
		Clasa pozitivă	Clasa(ele) negativă(e)
Rezultate calculate	Clasa pozitivă	<i>True positiv (TP)</i>	<i>False positiv (FP)</i>
	Clasa(ele) negativă(e)	<i>False negative (FN)</i>	<i>True negative (TN)</i>

Învățare supervizată – evaluare

Măsuri de performanță

□ Măsuri statistice

■ Funcția de loss

- Diferența între ieșirea dorită (D) și cea calculată (C)
- L_2 norm - Quadratic cost (*mean squared error*) $\sum || D - C ||^2$
- L_1 norm $\sum | D - C |$
- SVM loss (hinge loss, max-margin loss) $\sum_i \sum_{j, j \neq y_i} \max(C_j - D_{y_i} + \Delta, 0)$
- Softmax loss $\sum [- \ln(\exp(D) / \sum_{j, j \neq y_i} \exp(C_j))]$
- Cross-entropy $-\sum [D \ln C + (1 - D) \ln(1 - C)] / n$

Învățare supervizată – evaluare

Condiții fundamentale

- Distribuția datelor de antrenament și test este aceeași
 - În practică, o astfel de condiție este adesea violată
- Exemplele de antrenament trebuie să fie reprezentative pentru datele de test

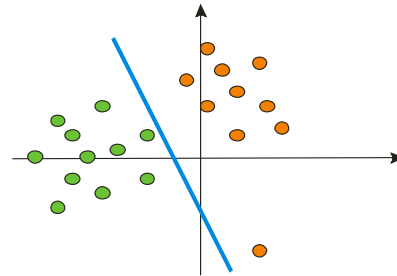
Învățare supervizată – tipologie

- După tipul de date de ieșire
 - Real → probleme de regresie
 - Etichete → probleme de clasificare (regresie logistică)
 - Clasificare binară
 - Ieșiri (output-uri) binare → nr binar de etichete posibile ($k = 2$)
 - Ex. diagnostic de cancer malign sau benign
 - Ex. email acceptat sau refuzat (spam)
 - Clasificare multi-clasă
 - Ieșiri multiple → nr > 2 de etichete posibile ($k > 2$)
 - Ex. recunoașterea cifrei 0, 1, 2, ... sau 9
 - Ex. risc de creditare mic, mediu, mare și foarte mare
 - Clasificare multi-etichetă
 - Fiecărei ieșiri îi pot corespunde una sau mai multe etichete
 - Ex. frumos → adjectiv, adverb

Învățare supervizată – tipologie

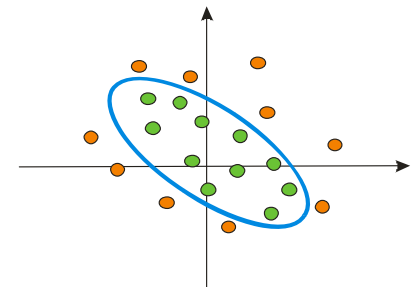
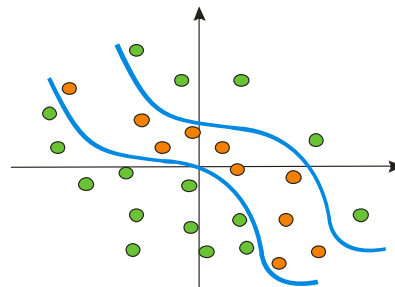
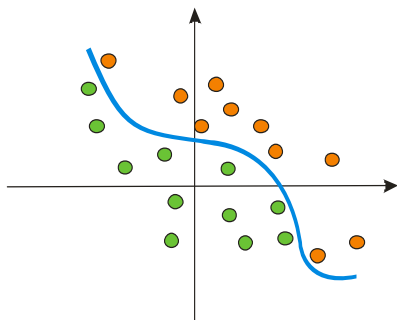
□ După forma clasificatorului

■ Clasificare liniară



■ Clasificare ne-liniară

- se crează o rețea de clasificatori liniari
- se mapează datele într-un spațiu nou (mai mare) unde ele devin separabile

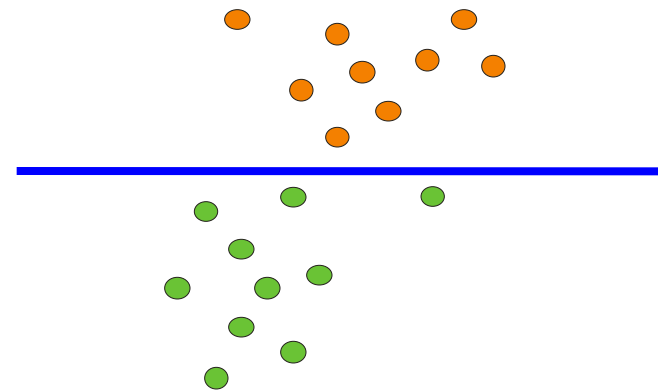


Învățare supervizată – tipologie

□ După caracteristicile datelor

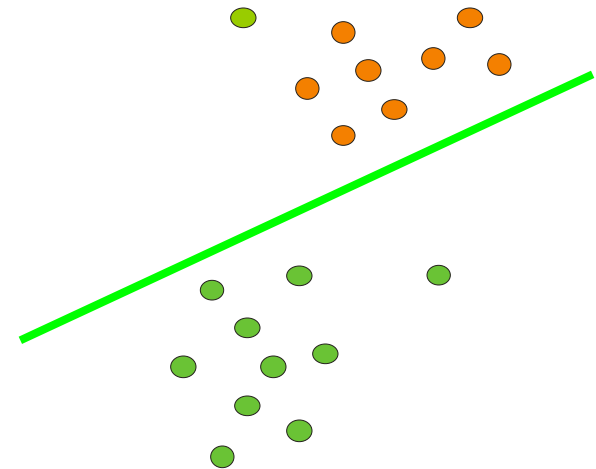
■ Clasificare pt date perfect separabile

- Clasificare fără eroare



■ Clasificare pt date ne-separabile

- Clasificare cu o anumită eroare (anumite date sunt plasate eronat în clase)



Învățare supervizată – tipologie

□ După algoritm

■ Bazată doar pe instanțe

- Folosește direct datele, fără a crea un model de separare
- Ex. algoritmul cel mai apropiat vecin (*k-nearest neighbour*)

■ Discriminative

- Estimează o separare al datelor
- Ex. arbori de decizie, rețele neuronale artificiale, mașini cu suport vectorial, algoritmi evolutivi

■ Generative

- Construiește un model probabilistic
- Ex. rețele Bayesiene

Învățare supervizată – algoritmi

- Cel mai apropiat vecin
 - Arbori de decizie
 - Sisteme bazate pe reguli
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi
- clasificare
- regresie

Învățare supervizată – algoritmi

Problemă de clasificare

□ Se dă

- un set de date (exemple, instanțe, cazuri)
 - date de antrenament – sub forma unor perechi ($\text{attribute_data}_i, \text{ieșire}_i$), unde
 - $i = 1, N$ ($N = \text{nr datelor de antrenament}$)
 - $\text{attribute_data}_i = (\text{atr}_{i1}, \text{atr}_{i2}, \dots, \text{atr}_{im})$, m – nr atributelor (caracteristicilor, proprietăților) unei date
 - $\text{ieșire}_i =$
 - o categorie dintr-o mulțime dată (predefinită) cu k elemente (k – nr de clase)
 - date de test – sub forma (attribute_data_i), $i = 1, n$ ($n = \text{nr datelor de test}$)

□ Să se determine

- o funcție (necunoscută) care realizează corespondența atribut – ieșire pe datele de antrenament
- ieșirea (clasa) asociată unei date (noi) de test folosind funcția învățată pe datele de antrenament

Învățare supervizată – algoritmi

Problemă de regresie

□ Se dă

- un set de date (exemple, instanțe, cazuri)
 - date de antrenament – sub forma unor perechi ($\text{attribute_data}_i, \text{ieșire}_i$), unde
 - $i = 1, N$ ($N = \text{nr datelor de antrenament}$)
 - $\text{attribute_data}_i = (\text{atr}_{i1}, \text{atr}_{i2}, \dots, \text{atr}_{im})$, $m = \text{nr atributelor (caracteristicilor, proprietăților) unei date}$
 - ieșire_i
 - un număr real
 - date de test – sub forma (attribute_data_i), $i = 1, n$ ($n = \text{nr datelor de test}$)

□ Să se determine

- o funcție (necunoscută) care realizează corespondența atribut – ieșire pe datele de antrenament
- Ieșirea (clasa/valoarea) asociată unei date (noi) de test folosind funcția învățată pe datele de antrenament

Învățare supervizată – algoritmi

Cel mai apropiat vecin (*k*-nearest neighbour)

- Cel mai simplu algoritm de clasificare
- În etapa de antrenament, algoritmul doar citește datele de intrare (atributele și clasa fiecărei instanțe)
- În etapa de testare, pentru o nouă instanță (fără clasă) se caută (printre instanțele de antrenament) cei mai apropiați *k* vecini și se preia clasa majoritară a acestor *k* vecini
- Căutarea vecinilor se bazează pe:
 - distanța Minkowski (Manhattan, Euclidiană) – attribute continue
 - distanța Hamming, Levensthein – analiza textelor
 - alte distanțe (funcții kernel)

Învățare supervizată – algoritmi

Arbori de decizie

□ Scop

- Divizarea unei colecții de articole în seturi mai mici prin aplicarea succesivă a unor reguli de decizie → adresarea mai multor întrebări
 - Fiecare întrebare este formulată în funcție de răspunsul primit la întrebarea precedentă
- Elementele se caracterizează prin informații non-metrică

□ Definiție

■ Arborele de decizie

- Un graf special → arbore orientat bicolor
- Conține noduri de 3 tipuri:
 - Noduri de decizie → posibilitățile decidentului (ex. Diversele examinări sau tratamente la care este supus pacientul) și indică un test pe un atribut al articolului care trebuie clasificat
 - Noduri ale hazardului – evenimente aleatoare în afara controlului decidentului (rezultatul examinărilor, efectul terapiei)
 - Noduri rezultat – situațiile finale cărora li se asociază o utilitate (apreciată aprioric de către un pacient generic) sau o etichetă
- Nodurile de decizie și cele ale hazardului alternează pe nivelele arborelui
- Nodurile rezultat – noduri terminale (frunze)
- Muchiile arborelui (arce orientate) → consecințele în timp (rezultate) ale deciziilor, respectiv ale realizării evenimentelor aleatoare (pot fi însoțite de probabilități)
- Fiecare nod intern corespunde unui atribut
- Fiecare ramură de sub un nod (atribut) corespunde unei valori a atributului
- Fiecare frunză corespunde unei clase (ieșire de tip discret)

Învățare supervizată – algoritmi

Arbori de decizie

□ Tipuri de probleme

- Exemplele (instanțele) sunt reprezentate printr-un număr fix de atribute, fiecare atribut putând avea un număr limitat de valori
- Funcția obiectiv ia valori de tip discret
- AD reprezintă o disjuncție de mai multe conjuncții
 - fiecare conjuncție fiind de forma atributul a_i are valoarea v_j
- Datele de antrenament pot conține erori
- Datele de antrenament pot fi incomplete
 - Anumitor exemple le pot lipsi valorile pentru unele atribute

- Probleme de clasificare
 - Binară
 - exemple date sub forma [(atribut_{ij}, valoare_{ij}), clasă_i, $i=1,2,\dots,n$, $j=1,2,\dots,m$, clasă_i putând lua doar 2 valori]
 - Multi-clasă
 - exemple date sub forma [(atribut_{ij}, valoare_{ij}), clasă_i, $i=1,2,\dots,n$, $j=1,2,\dots,m$, clasă_i putând lua doar k valori]

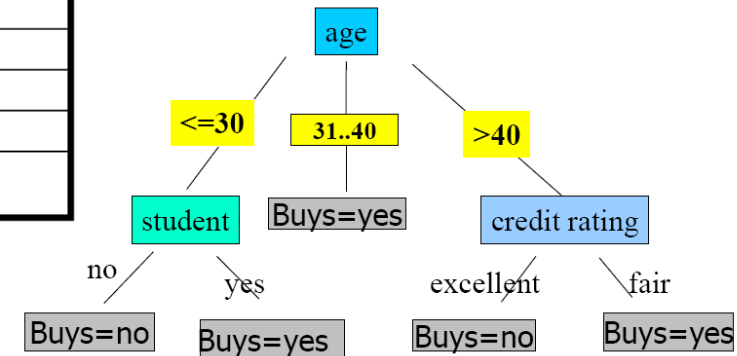
- Probleme de regresie
 - AD se construiesc similar cazului problemei de clasificare, dar în locul etichetării fiecărui nod cu eticheta unei clase se asociază nodului o valoare reală sau o funcție dependentă de intrările nodului respectiv
 - Spațiul de intrare se împarte în regiuni de decizie prin tăieturi paralele cu axele Ox și Oy
 - Are loc o transformare a ieșirilor discrete în funcții continue
 - Calitatea rezolvării problemei
 - Eroare (pătratică sau absolută) de predicție

Învățare supervizată – algoritmi

Arbori de decizie

□ Exemplu

rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No



Învățare supervizată – algoritmi

Arbori de decizie

□ Proces

- Construirea (creșterea, inducția) arborelui
 - Se bazează pe un set de date de antrenament
 - Lucrează de jos în sus sau de sus în jos (prin divizare – *splitting*)
- Utilizarea arborelui ca model de rezolvare a problemelor
 - Ansamblul deciziilor efectuate de-a lungul unui drum de la rădăcină la o frunză formează o regulă
 - Regulile formate în AD sunt folosite pentru etichetarea unor noi date
- Tăierea (curățirea) arborelui (pruning)
 - Se identifică și se mută/elimină ramurile care reflectă zgomote sau excepții

Învățare supervizată – algoritmi

Arbori de decizie

□ Proces

■ Construirea AD

□ Exemplu

rec	Age	Income	Student	Credit_rating	Buys_computer(CLASS)
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

Învățare supervizată – algoritmi

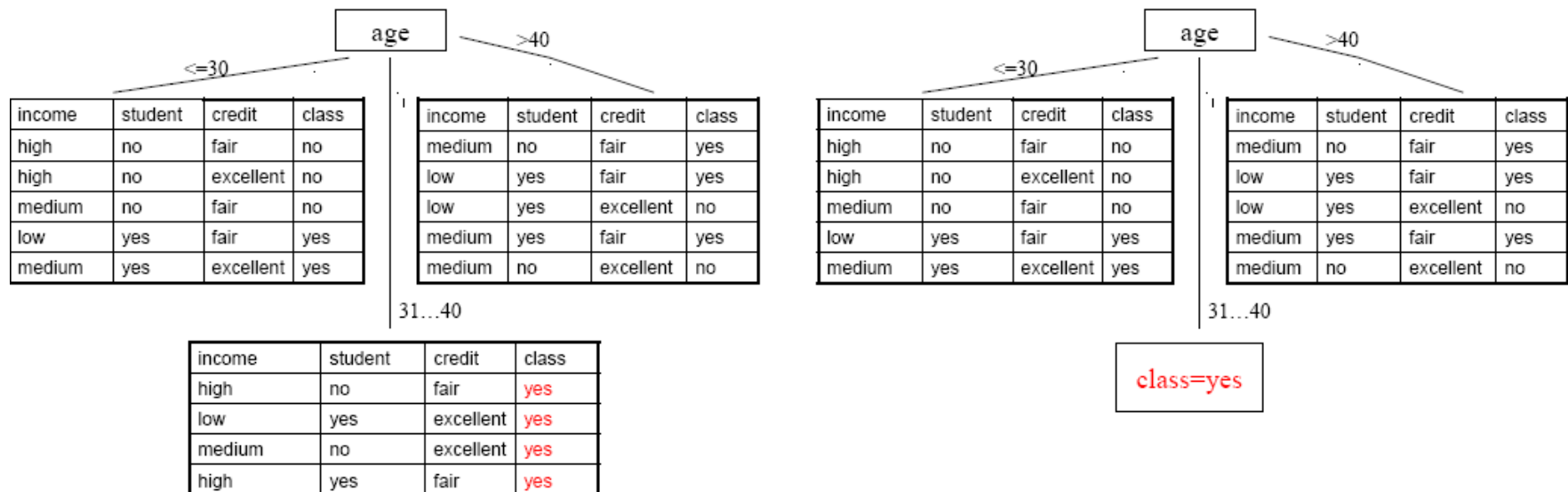
Arbori de decizie

□ Proces

■ Construirea AD

□ Exemplu

- Pentru rădăcină se alege atributul *age*



Învățare supervizată – algoritmi

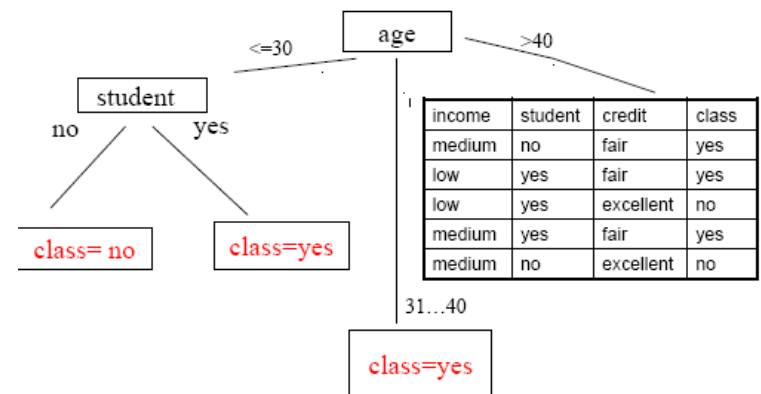
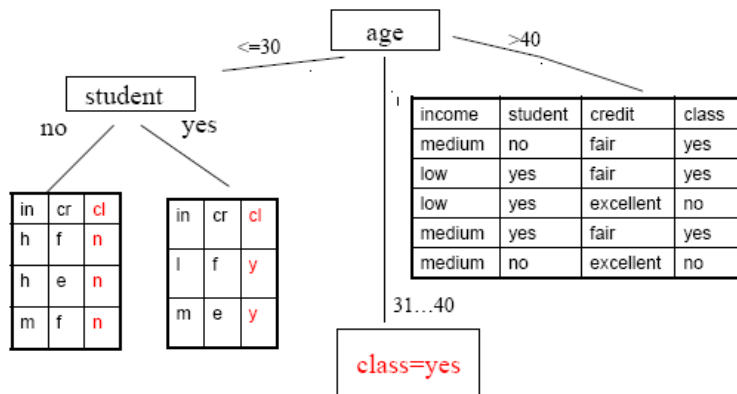
Arbori de decizie

□ Proces

■ Construirea AD

□ Exemplu

- Pentru rădăcină se alege atributul *age*
- Pe ramura ≤ 30 se alege atributul *student*



Învățare supervizată – algoritmi

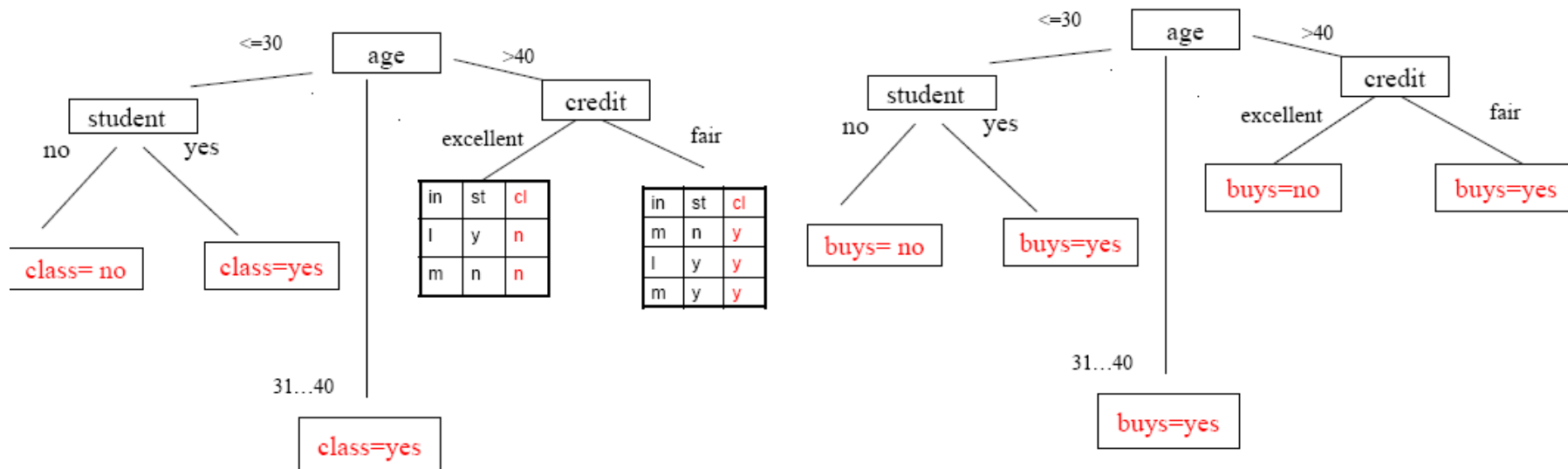
Arbori de decizie

□ Proces

■ Construirea AD

□ Exemplu

- Pentru rădăcină se alege atributul *age*
- Pe ramura ≤ 30 se alege atributul *student*
- Pe ramura > 40 se alege atributul *credit*



Învățare supervizată – algoritmi

Arbori de decizie

□ Proces

- Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut → Câștigul de informație

□ O măsură de impuritate

- 0 (minimă) dacă toate exemplele aparțin aceeași clase
- 1 (maximă) dacă avem număr egal de exemple din fiecare clasă

□ Se bazează pe entropia datelor

- măsoară impuritatea datelor
- numărul sperat (așteptat) de biți necesari pentru a coda clasa unui element oarecare din setul de date
- clasificare binară (cu 2 clase): $E(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$, unde
 - p_+ - proporția exemplelor pozitive în setul de date S
 - p_- - proporția exemplelor negative în setul de date S
- clasificare cu mai multe clase: $E(S) = \sum_{i=1, 2, \dots, k} -p_i \log_2 p_i$ - entropia datelor relativ la atributul țintă (atributul de ieșire), unde
 - p_i - proporția exemplelor din clasa i în setul de date S

□ câștigul de informație (*information gain*) al unei caracteristici a (al unui atribut a) datelor

- Reducerea entropiei setului de date ca urmare a eliminării atributului a
- $Gain(S, a) = E(S) - \sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$
- $\sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$ - *informația scontată*

Învățare supervizată – algoritmi

Arbori de decizie

□ Proces

- Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut → Câștigul de informație
 - exemplu

	a1	a2	a3	Clasa
d1	mare	roșu	cerc	clasa 1
d2	mic	roșu	pătrat	clasa 2
d3	mic	roșu	cerc	clasa 1
d4	mare	albastru	cerc	clasa 2

$$S = \{d1, d2, d3, d4\} \rightarrow p_+ = 2 / 4, p_- = 2 / 4 \rightarrow E(S) = - p_+ \log_2 p_+ - p_- \log_2 p_- = 1$$

$$S_{v=mare} = \{d1, d4\} \rightarrow p_+ = 1/2, p_- = 1/2 \rightarrow E(S_{v=mare}) = 1$$

$$S_{v=mic} = \{d2, d3\} \rightarrow p_+ = 1/2, p_- = 1/2 \rightarrow E(S_{v=mic}) = 1$$

$$S_{v=roșu} = \{d1, d2, d3\} \rightarrow p_+ = 2/3, p_- = 1/3 \rightarrow E(S_{v=roșu}) = 0.923$$

$$S_{v=albastru} = \{d4\} \rightarrow p_+ = 0, p_- = 1 \rightarrow E(S_{v=albastru}) = 0$$

$$S_{v=cerc} = \{d1, d3, d4\} \rightarrow p_+ = 2/3, p_- = 1/3 \rightarrow E(S_{v=cerc}) = 0.923$$

$$S_{v=patrat} = \{d2\} \rightarrow p_+ = 0, p_- = 1 \rightarrow E(S_{v=patrat}) = 0$$

$$\text{Gain}(S, a) = E(S) - \sum_{v \in \text{valori}(a)} |S_v| / |S| E(S_v)$$

$$\text{Gain}(S, a_1) = 1 - (|S_{v=mare}| / |S| E(S_{v=mare}) + |S_{v=mic}| / |S| E(S_{v=mic})) = 1 - (2/4 * 1 + 2/4 * 1) = 0$$

$$\text{Gain}(S, a_2) = 1 - (|S_{v=roșu}| / |S| E(S_{v=roșu}) + |S_{v=albastru}| / |S| E(S_{v=albastru})) = 1 - (3/4 * 0.923 + 1/4 * 0) = 0.307$$

$$\text{Gain}(S, a_3) = 1 - (|S_{v=cerc}| / |S| E(S_{v=cerc}) + |S_{v=patrat}| / |S| E(S_{v=patrat})) = 1 - (3/4 * 0.923 + 1/4 * 0) = 0.307$$

Învățare supervizată – algoritmi

Arbori de decizie

□ Proces

- Construirea AD → Algoritmul ID3/C4.5 → Selectare atribut → Rata câștigului
 - Penalizează un atribut prin încorporarea unui termen – split information – sensibil la gradul de împrăștiere și uniformitate în care atributul separă datele
 - Split information – entropia relativ la valorile posibile ale atributului a
 - S_v – proporția exemplilor din setul de date S care au atributul a egal cu valoarea v
 - $\text{splitInformation}(S,a) = - \sum_{v=\text{value}(a)} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$

Învățare supervizată – algoritmi

Arbori de decizie

□ Proces

■ Construirea arborelui

■ Utilizarea arborelui ca model de rezolvare a problemelor

□ Ideea de bază

- Se extrag regulile formate în arborele anterior construit → Reguli extrase din arborele dat în exemplul anterior:
 - *IF age = " ≤ 30 " AND student = "no" THEN buys_computer = "no"*
 - *IF age = " ≤ 30 " AND student = "yes" THEN buys_computer = "yes"*
 - *IF age = "31...40" THEN buys_computer = "yes"*
 - *IF age = " > 40 " AND credit_rating = "excellent" THEN buys_computer = "no"*
 - *IF age = " > 40 " AND credit_rating = "fair" THEN buys_computer = "yes"*
- Regulile sunt folosite pentru a clasifica datele de test (date noi). Fie x o dată pentru care nu se știe clasa de apartenență → Regulile se pot scrie sub forma unor predicate astfel:
 - *IF age(x , ≤ 30) AND student(x , no) THEN buys_computer (x , no)*
 - *IF age(x , ≤ 30) AND student (x , yes) THEN buys_computer (x , yes)*

□ Dificultăți

- Underfitting (sub-potrivire) → AD indus pe baza datelor de antrenament este prea simplu → eroare de clasificare mare atât în etapa de antrenare, cât și în cea de testare
- Overfitting (supra-potrivire, învățare pe derost) → AD indus pe baza datelor de antrenament se potrivește prea accentuat cu datele de antrenament, nefiind capabil să generalizeze pentru date noi
- Soluții:
 - fasonarea arborelui (pruning) → Îndepărtarea ramurilor nesemnificative, redundante → arbore mai puțin stufos
 - Validare cu încrucișare

Învățare supervizată – algoritmi

Arbori de decizie

□ Proces

- Construirea arborelui
- Utilizarea arborelui ca model de rezolvare a problemelor
- Tăierea (fasonarea) arborelui

□ Necesitate

- Odată construit AD, se pot extrage reguli (de clasificare) din AD pentru a putea reprezenta cunoștințele sub forma regulilor dacă-atunci atât de ușor de înțeles de către oameni
- O regulă este creată (extrasă) prin parcurgerea AD de la rădăcină până la o frunză
- Fiecare pereche (atribut, valoare), adică (nod, muchie), formează o conjuncție în ipoteza regulii (partea dacă)
 - Mai puțin ultimul nod din drumul parcurs care este o frunză și reprezintă consecința (ieșirea, partea atunci) regulii

□ Tipologie

- Prealabilă (pre-pruning)
 - Se oprește creșterea arborelui în timpul inducției prin sistarea divizării unor noduri care devin astfel frunze etichetate cu clasa majoritară a exemplorilor aferente nodului respectiv
- Ulterioară (post-pruning)
 - După ce AD a fost creat (a crescut) se elimină ramurile unor noduri care devin astfel frunze → se reduce eroarea de clasificare (pe datele de test)

Învățare supervizată – algoritmi

Arbori de decizie

□ Tool-uri

- <http://webdocs.cs.ualberta.ca/~aixplore/learning/DecisionTrees/Applet/DecisionTreeApplet.html>
- WEKA → J48
- <http://id3alg.altervista.org/>
- <http://www.rulequest.com/Personal/c4.5r8.tar.gz>

□ Biblio

- <http://www.public.asu.edu/~kirkwood/DASstuff/decisiontrees/index.html>

Învățare supervizată – algoritmi

Arbori de decizie

□ Avantaje

- Ușor de înțeles și interpretat
- Permit utilizarea datelor nominale și categoriale
- Logica deciziei poate fi urmărită ușor, regulile fiind vizibile
- Lucrează bine cu seturi mari de date

□ Dezavantaje

- Instabilitate → modificarea datelor de antrenament
- Complexitate → reprezentare
- Greu de manevrat
- Costuri mari pt inducerea AD
- Inducerea AD necesită multă informație

Învățare supervizată – algoritmi

Arbori de decizie

□ Dificultăți

■ Existența mai multor arbori

- Cât mai mici
- Cu o acuratețe cât mai mare (ușor de "citit" și cu performanțe bune)
- Găsirea celui mai bun arbore → problemă NP-dificilă

■ Alegerea celui mai bun arbore

- Algoritmi euristici
- ID3 → cel mai mic arbore acceptabil
 - → teorema lui Occam: "always choose the simplest explanation"

■ Attribute continue

- Separarea în intervale
 - Câte intervale?
 - Cât de mari sunt intervalele?

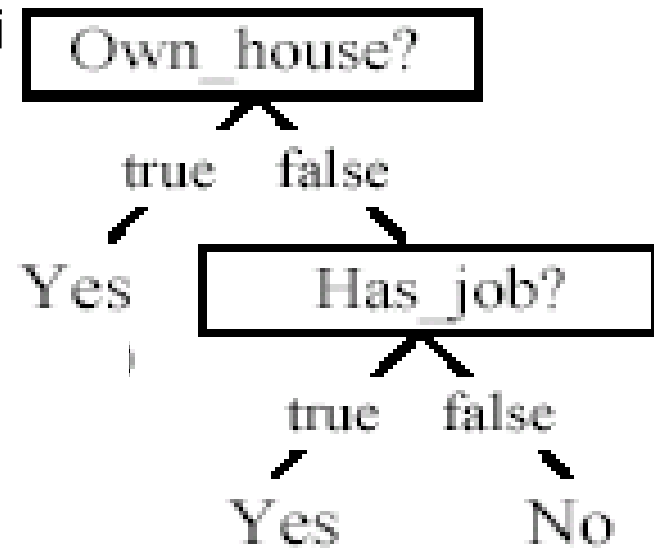
■ Arbori prea adânci sau prea stufoși

- Fasonarea prealabilă (pre-pruning) → oprirea construirii arborelui mai devreme
- Fasonarea ulterioară (post-pruning) → înlăturarea anumitor ramuri

Învățare supervizată – algoritmi

Sisteme bazate pe reguli

- ❑ Transformarea AD într-un set de reguli
- ❑ Fiecare drum din arbore → regulă
- ❑ Regulile *IF-THEN* pot fi identificate din date



Own_house = true → Class = Yes

Own_house = false, Has_job = true → Class = Yes

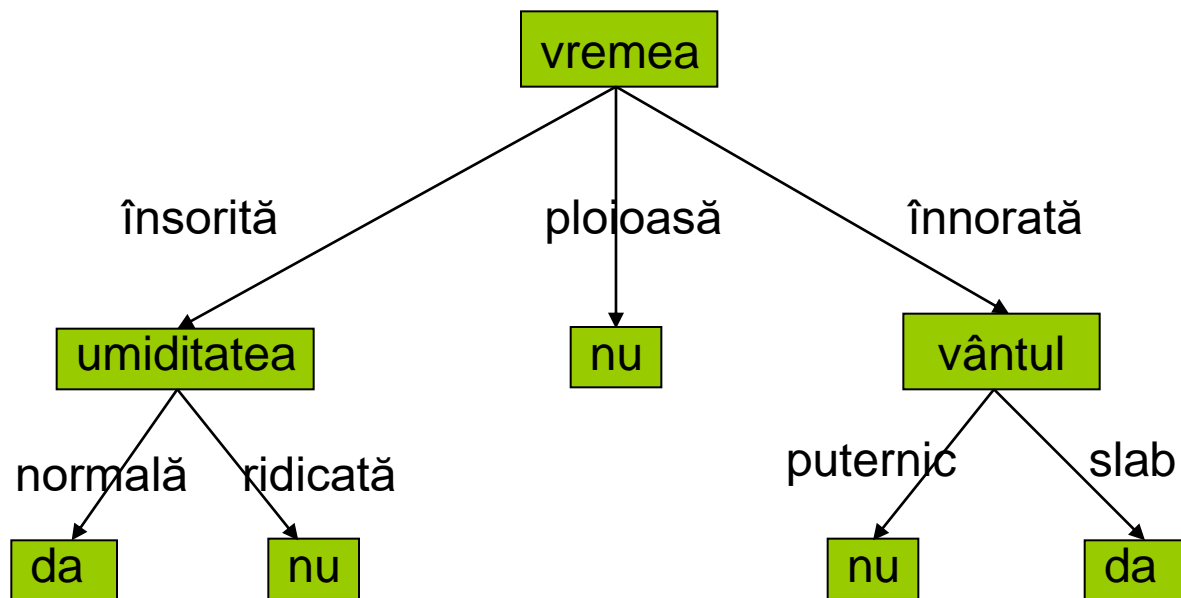
Own_house = false, Has_job = false → Class = No

Învățare supervizată – algoritmi

Sisteme bazate pe reguli

Conversia unui AD în reguli

- Vremea propice pentru tenis



- DACĂ vremea=înnorată și vântul=slab ATUNCI joc tenis

Învățare supervizată – algoritmi

Sisteme bazate pe reguli

- Secvențele de reguli = liste de decizii
- Găsirea regulilor
 - Acoperire secvențială
 - Se învață o regulă
 - Se elimină exemplele care respectă regula
 - Se caută noi reguli

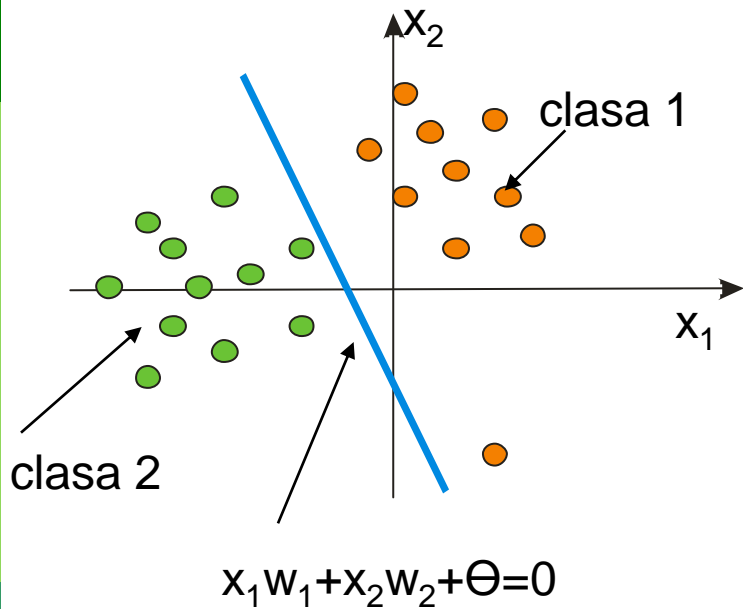
Învățare supervizată – algoritmi

Clasificare

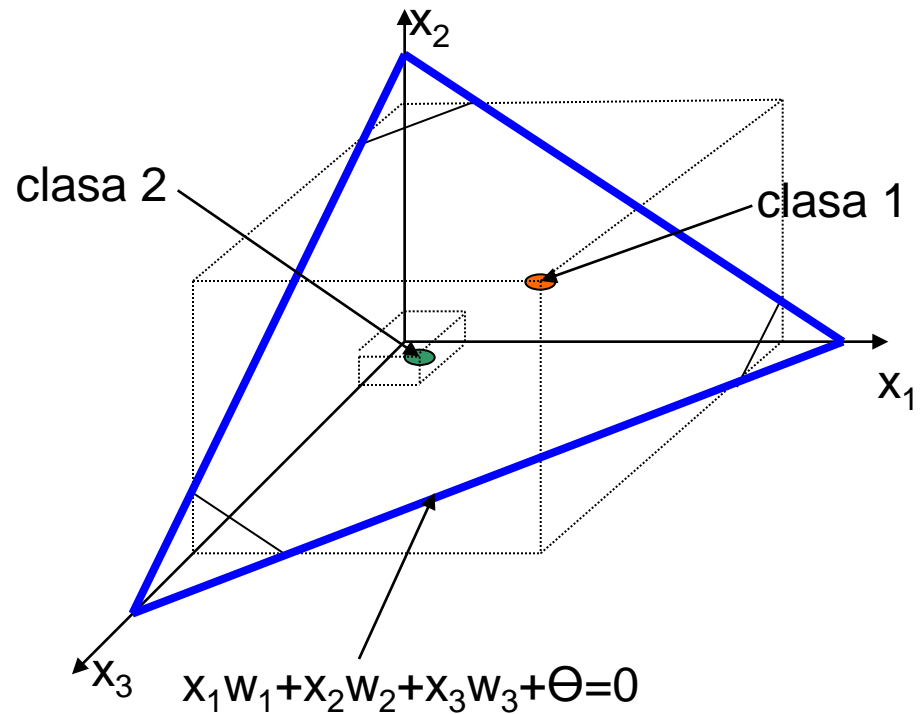
- Clasificare binară pt orice fel de date de intrare (discrete sau continue)
 - Datele pot fi separate de:
 - o dreaptă $\rightarrow ax + by + c = 0$ (dacă $m = 2$)
 - un plan $\rightarrow ax + by + cz + d = 0$ (dacă $m = 3$)
 - un hiperplan $\sum a_i x_i + b = 0$ (dacă $m > 3$)
 - Cum găsim modelul de separare (valorile optime pt. a, b, c, d, a_i și forma modelului)?
 - Rețele neuronale artificiale
 - Mașini cu suport vectorial
 - Algoritmi evolutivi

Învățare supervizată – algoritmi

Rețele neuronale artificiale



Clasificare binară cu $m=2$ intrări

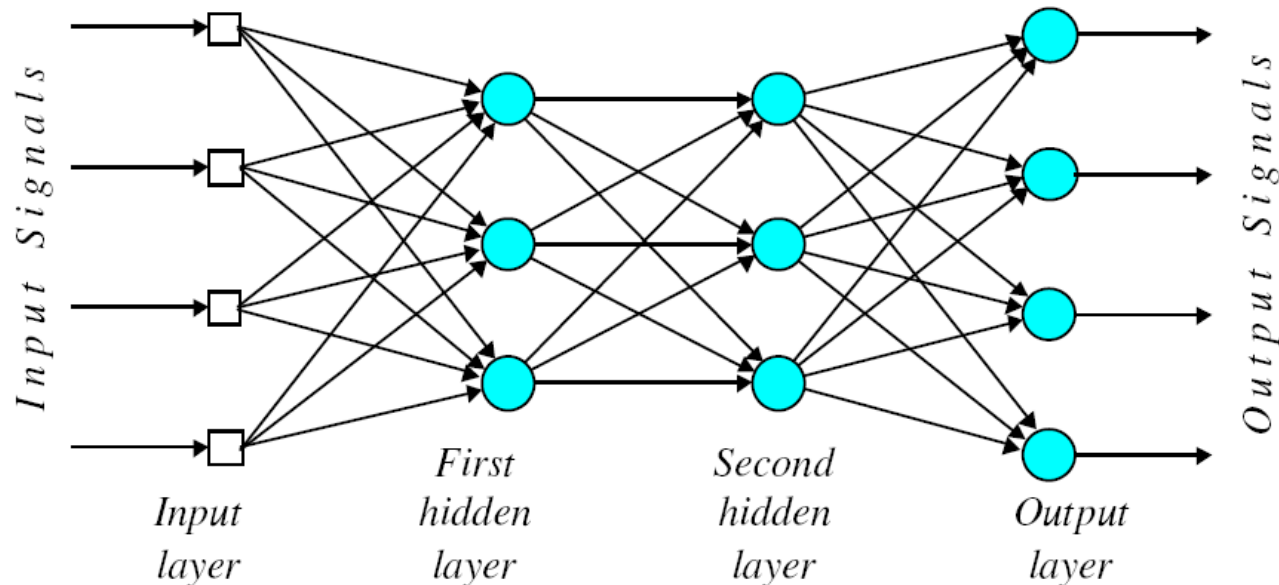


Clasificare binară cu $m=3$ intrări

Învățare supervizată – algoritmi

Rețele neuronale artificiale

- Similar unei rețele neuronale biologice
- O mulțime de neuroni dispuși ca într-un graf (un nod \rightarrow un neuron) pe mai multe straturi (*layer*)
 - Strat de intrare
 - Conține m (nr de attribute al unei date) noduri
 - Strat de ieșire
 - Conține r (nr de ieșiri) noduri
 - Straturi intermediare (ascunse) – rol în “complicarea” rețelei
 - Diferite structuri
 - Diferite mărimi



Învățare supervizată – algoritmi

Rețele neuronale artificiale

□ Cum învață rețeaua?

- Plecând de la un set de n date de antrenament de forma $((x_{p1}, x_{p2}, \dots, x_{pm}, y_{p1}, y_{p2}, \dots, y_{pr}))$ cu $p = 1, 2, \dots, n$, m – nr atributelor, r – nr ieșirilor
- Se formează o RNA cu m noduri de intrare, r noduri de ieșire și o anumită structură internă (un anumit nr de nivele ascunse, fiecare nivel cu un anumit nr de neuroni)
- Se caută valorile optime ale ponderilor între oricare 2 noduri ale rețelei prin minimizarea erorii (diferența între outputul real y și cel calculat de către rețea)
- Rețeaua = mulțime de unități primitive de calcul interconectate între ele
→
 - Învățarea rețelei = \cup învățarea unităților primitive (unitate liniară, unitate sigmoidală, etc)

Învățare supervizată – algoritmi

Rețele neuronale artificiale

- Neuronul ca element simplu de calcul
 - Structura neuronului
 - Fiecare nod are intrări și ieșiri
 - Fiecare nod efectuează un calcul simplu prin intermediul unei funcții asociate
 - Procesarea neuronului
 - Se transmite informația neuronului → se calculează suma ponderată a intrărilor
 - Neuronul procesează informația → se folosește o funcție de activare
 - Funcția semn → perceptron
 - Funcția liniară → unitate liniară
 - Funcția sigmoidală → unitate sigmoidală
 - Se citește răspunsul neuronului → se stabilește dacă rezultatul furnizat de neuron coincide sau nu cu cel dorit (real)
 - Învățarea neuronului – algoritmul de învățare a ponderilor care procesează corect informațiile
 - Se pornește cu un set inițial de ponderi oarecare
 - Cât timp nu este îndeplinită o condiție de oprire
 - Se stabilește calitatea ponderilor curente
 - Se modifică ponderile astfel încât să se obțină rezultate mai bune

Învățare supervizată – algoritmi

Rețele neuronale artificiale

□ Procesare → Funcția de activare

- Funcția constantă $f(\text{net}) = \text{const}$

- Funcția prag (c - pragul)

$$f(\text{net}) = \begin{cases} a, & \text{dacă } \text{net} < c \\ b, & \text{dacă } \text{net} > c \end{cases}$$

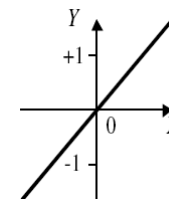
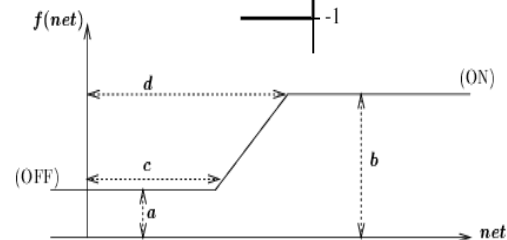
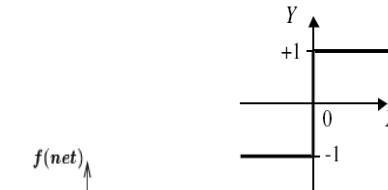
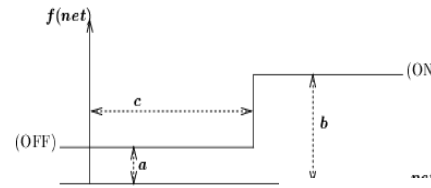
- Pentru $a=+1$, $b=-1$ și $c=0$ → funcția semn
- Funcție discontinuă

- Funcția rampă

$$f(\text{net}) = \begin{cases} a, & \text{dacă } \text{net} \leq c \\ b, & \text{dacă } \text{net} \geq d \\ a + \frac{(\text{net}-c)(b-a)}{d-c}, & \text{altfel} \end{cases}$$

- Funcția liniară $f(\text{net})=a*\text{net} + b$

- Pentru $a = 1$ și $b = 0$ → funcția identitate $f(\text{net})=\text{net}$
- Funcție continuă



Învățare supervizată – algoritmi

Rețele neuronale artificiale

□ Procesare → Funcția de activare

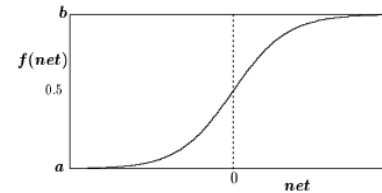
■ Funcția sigmoidală

- În formă de S
- Continuă și diferentiabilă în orice punct
- Simetrică rotațional față de un anumit punct ($net = c$)
- Atinge asimptotic puncte de saturație

$$\lim_{net \rightarrow -\infty} f(net) = a \qquad \lim_{net \rightarrow \infty} f(net) = b$$

- Exemple de funcții sigmoidale:

$$f(net) = z + \frac{1}{1 + \exp(-x \cdot net + y)}$$
$$f(net) = \tanh(x \cdot net - y) + z$$



unde $\tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$

- Pentru $y=0$ și $z = 0 \rightarrow a=0, b = 1, c=0$
- Pentru $y=0$ și $z = -0.5 \rightarrow a=-0.5, b = 0.5, c=0$
- Cu cât x este mai mare, cu atât curba este mai abruptă

Învățare supervizată – algoritmi

Rețele neuronale artificiale

□ Procesare → Funcția de activare

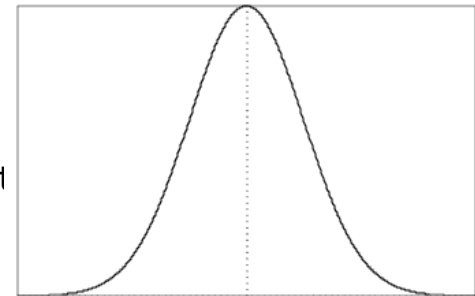
■ Funcția Gaussiană

- În formă de clopot
- Continuă
- Atinge asimptotic un punct de saturație

$$\lim_{net \rightarrow \infty} f(net) = a$$

- Are un singur punct de optim (maxim) – atins când net
- Exemplu

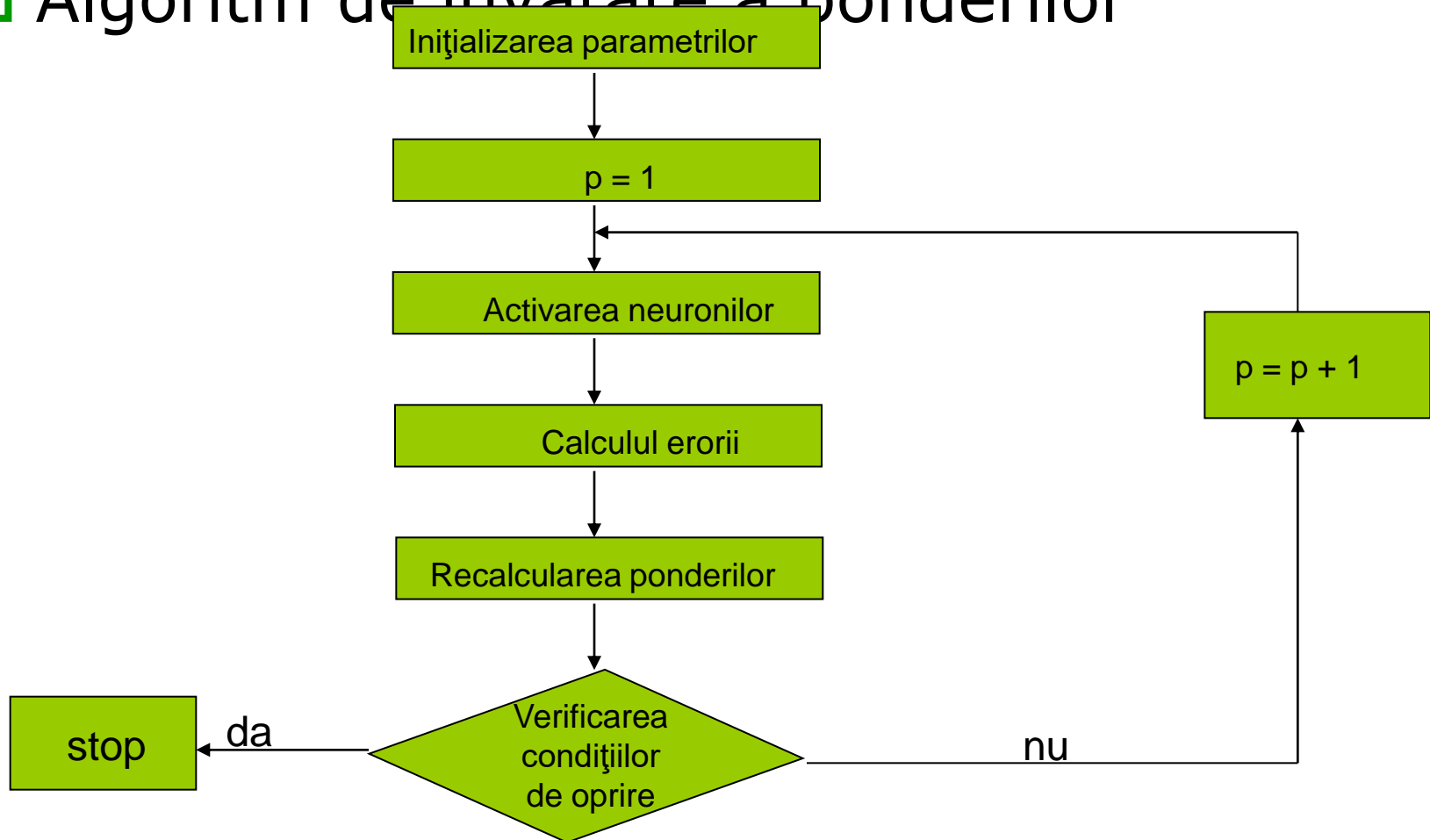
$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{net-\mu}{\sigma}\right)^2\right]$$



Învățare supervizată – algoritmi

Rețele neuronale artificiale

□ Algoritm de învățare a ponderilor



Învățare supervizată – algoritmi

Rețele neuronale artificiale

□ Învățare

■ 2 reguli de bază

□ Regula perceptronului

1. Se porneste cu un set de ponderi oarecare
2. Se stabilește calitatea modelului creat pe baza acestor ponderi pentru una dintre datele de intrare
3. Se ajustează ponderile în funcție de calitatea modelului
4. Se reia algoritmul de la pasul 2 până când se ajunge la calitate maximă

□ Regula Delta

- Similar regulii perceptronului dar calitatea unui model se stabilește în funcție de toate datele de intrare (tot setul de antrenament)

Învățare supervizată – algoritmi

Rețele neuronale artificiale

Perceptron - exemplu

Epoch	Inputs		Desired output Y_d	Initial weights		Actual output Y	Error e	Final weights	
	x_1	x_2		w_1	w_2			w_1	w_2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	-1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	-1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

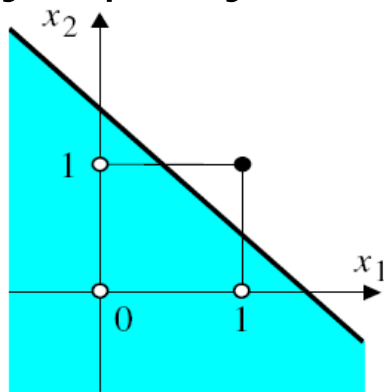
Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Învățare supervizată – algoritmi

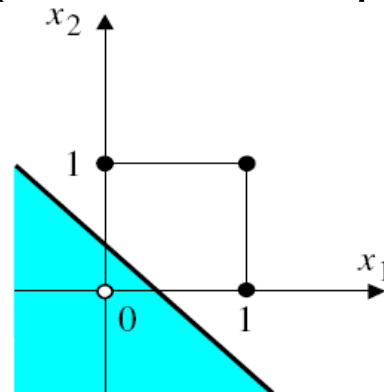
Rețele neuronale artificiale

Perceptron - limitări

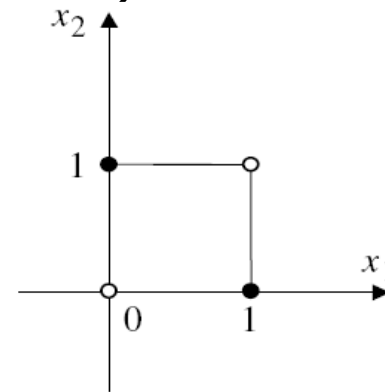
- Un perceptron poate învăța operațiile AND și OR, dar nu poate învăța operația XOR (nu e liniar separabilă)



(a) AND ($x_1 \cap x_2$)



(b) OR ($x_1 \cup x_2$)



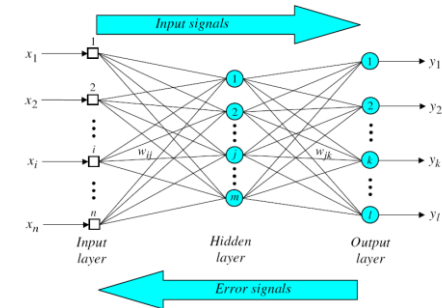
(c) Exclusive-OR
($x_1 \oplus x_2$)

- Nu poate clasifica date non-liniar separabile
 - soluția = mai mulți neuroni

Învățare supervizată – algoritmi

Rețele neuronale artificiale

- ❑ Cum învață rețeaua cu mai mulți neuroni așezați pe unul sau mai multe straturi ?
 - RNA este capabilă să învețe un model mai complicat (nu doar liniar) de separare a datelor
 - Algoritmul de învățare a ponderilor → backpropagation
 - ❑ Bazat pe algoritmul scădere după gradient (versiunea clasică sau cea stocastică)
 - ❑ Îmbogățit cu:
 - Informația se propagă în RNA înainte (dinspre stratul de intrare spre cel de ieșire)
 - Eroarea se propagă în RNA înapoi (dinspre stratul de ieșire spre cel de intrare)
 - Pp că avem un set de date de antrenament de forma:
 - ❑ (x^d, t^d) , cu:
 - $x^d \in \mathbf{R}^m \rightarrow x^d = (x^d_1, x^d_2, \dots, x^d_m)$
 - $t^d \in \mathbf{R}^R \rightarrow t^d = (t^d_1, t^d_2, \dots, t^d_R)$
 - cu $d = 1, 2, \dots, n$
 - Presupunem 2 cazuri de RNA
 - ❑ O RNA cu un singur strat ascuns cu H neuroni → RNA₁
 - ❑ O RNA cu p straturi ascunse, fiecare strat cu H_i ($i = 1, 2, \dots, p$) neuroni → RNA_p



Învățare supervizată – algoritmi

Rețele neuronale artificiale

Algoritmul backpropagation

- Se inițializează ponderile
- Cât timp nu este îndeplinită condiția de oprire
 - Pentru fiecare exemplu (x^d, t^d)
 - Se activează fiecare neuron al rețelei
 - Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei
 - Se ajustează ponderile
 - Se stabilește și se propagă eroarea înapoi
 - Se stabilesc erorile corespunzătoare neuronilor din stratul de ieșire
 - Se propagă aceste erori înapoi în toată rețeaua → se distribuie erorile pe toate conexiunile existente în rețea proporțional cu valorile ponderilor asociate acestor conexiuni
 - Se modifică ponderile

Învățare supervizată – algoritmi

Rețele neuronale artificiale

- Condiții de oprire
 - S-a ajuns la eroare 0
 - S-au efectuat un anumit număr de iterații
 - La o iterație se procesează un singur exemplu
 - n iterații = o epocă

Învățare supervizată – algoritmi

Rețele neuronale artificiale (clasice)

- Architectures – special graphs with nodes placed on layers
 - Layers
 - Input layer – size = input's size (#features)
 - Hidden layers – various sizes (#layers, # neurons/layer)
 - Output layers – size = output size (e.g. # classes)
 - Topology
 - Full connected layers (one-way connections, recurrent connections)

- Mechanism
 - Neuron activation
 - Constant, step, linear, sigmoid
 - Cost & Loss function → smooth cost function (depends on w&b)
 - Difference between desired (D) and computed © output
 - Quadratic cost (*mean squared error*)
 - $\sum \|D - C\|^2 / 2n$
 - Cross-entropy
 - $-\sum [D \ln C + (1 - D) \ln(1 - C)] / n$
 - Learning algorithm
 - Perceptron rule
 - Delta rule (Simple/Stochastic Gradient Descent)

Învățare supervizată – algoritmi

Rețele neuronale artificiale (Convolutional Neural Networks)

- More layers (< 10)

- Wide NNs

- More nodes/layer

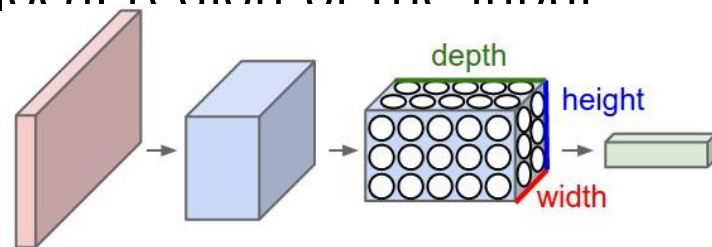
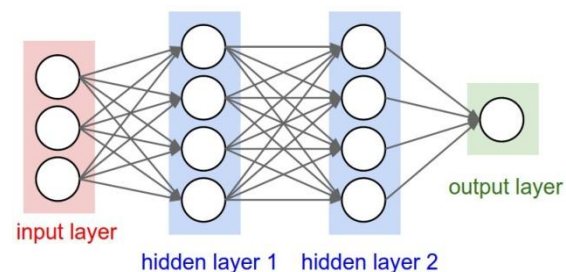
- Topology of connections

- Regular NNs \rightarrow fully connected
- Conv NNs \rightarrow partially connected

- connect each neuron to only a local region of the input volume

- Topology of layers

- Regular NNs \rightarrow linear layers
- Conv NNs \rightarrow 2D/3D layers (width, height, depth)



Conv NNs

- Layers of a Conv NN
 - Convolutional Layer → feature map
 - Convolution
 - Activation (thresholding)
 - Pooling/Aggregation Layer → size reduction
 - Fully-Connected Layer → answer

Conv NNs

□ Convolutional layer

■ Aim

- *learn data-specific kernels*

■ Filters or Local receptive fields or Kernels

□ content

- a little (square) window on the input pixels

□ How it works?

- slide the local receptive field across the entire input image

□ Size

- Size of field/filter (F)
- Stride (S)

□ Learning process

- each hidden neuron has
 - $F \times F$ shared weights connected to its local receptive field
 - a shared bias
 - an activation function
- each connection learns a weight
- the hidden neuron learns an overall bias as well
- all the neurons in the first hidden layer detect exactly the same feature (just at different locations in the input image) → map from input to the first hidden layer = feature map / activation map

Conv NNs

Convolutional Layer – How does it work?

- Take an input I (example, instance, data) of various dimensions

- A signal \rightarrow 1D input (I_{length})
- a grayscale image \rightarrow 2D input (I_{width} & I_{height})
- an RGB image \rightarrow 3D input (I_{width} , I_{height} & $I_{\text{depth}} = 3$)

- Consider a set of filters (kernels) $F_1, F_2, \dots, F_{\# \text{filters}}$

- A filter must have the same # dimensions as the input

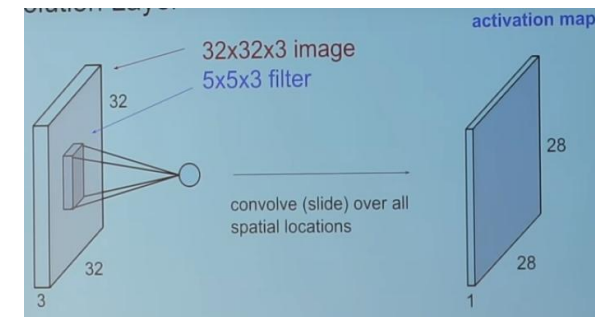
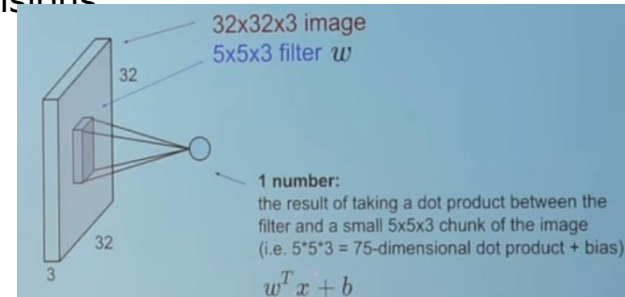
- A signal \rightarrow 1D filter
 - $F_{\text{length}} \ll I_{\text{length}}$
- a grayscale image \rightarrow 2D filter
 - $F_{\text{width}} \ll I_{\text{width}}$ & $F_{\text{height}} \ll I_{\text{height}}$
- an RGB image \rightarrow 3D filter
 - $F_{\text{width}} \ll I_{\text{width}}$ & $F_{\text{height}} \ll I_{\text{height}}$ & $F_{\text{depth}} = I_{\text{depth}} = 3$

- Apply each filter over the input

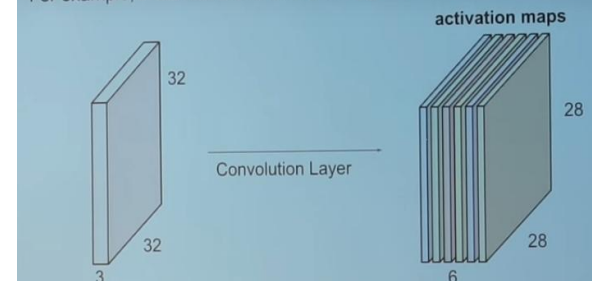
- Overlap filter over a window of the input
 - Stride
 - Padding
- Multiply the filter and the window
- Store the results in an activation map
 - # activation maps = # filters

- Activate all the elements of each activation map

- ReLU or other activation function



For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

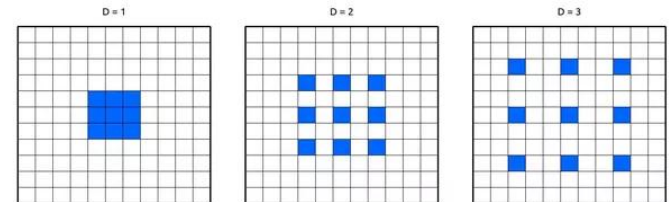


We stack these up to get a "new image" of size 28x28x6!

Conv NNs

□ Convolutional layer

- CONV layer's parameters consist of a set of learnable filters
 - Contiguous filters (without spaces between each cell)
 - Dilated filters (with spaces between each cell)
 - apply the same filter at different ranges using different dilation factors



- By sliding a filter → a 2-dimensional activation map
 - All filters → depth 2D activation maps
- Partial connected neurons
 - The spatial extent of this connectivity is a hyperparameter called the **receptive field** of the neuron (filter size)
 - The connections are local in space (along width and height), but always full along the entire depth of the input volume

Conv NNs

□ Convolutional layer - Hyperparameters

- input volume size N (L or $W \& H$ or $W \& H \& D$)
- size of zero-padding of input volume P (P_L or $P_W \& P_H$ or $P_W \& P_H$)
- the receptive field size (filter size) F (F_L , $F_W \& F_H$, $F_W \& F_H \& F_D$)
- stride of the convolutional layer S (S_L , $S_W \& S_H$, $S_W \& S_H$)
- # of filters (K)
 - depth of the output volume

- # neurons of an activation map = $(N + 2P - F)/S + 1$

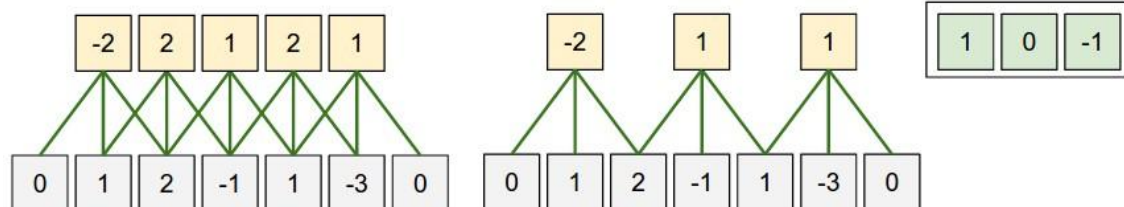
■ Output size

- $K * [(N + 2P - F)/S + 1]$

- $N = L = 5, P = 1,$

- $F = 3, S = 1$

- $F = 3, S = 2$



Conv NNs

- Convolutional layer - ImageNet challenge in 2012 (Alex Krizhevsky <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>)
 - Input images of size [227x227x3]
 - $F=11, S=4, P=0, K = 96 \rightarrow$ Conv layer output volume of size [55x55x96]
 - $55*55*96 = 290,400$ neurons in the first Conv Layer
 - each has $11*11*3 = 363$ weights and 1 bias.
 - $290400 * 364 = 105,705,600$ parameters on the first layer

Conv NNs

□ Convolutional layer - parameter sharing scheme

- constrain the neurons in each depth slice to use the same weights and bias
 - detect exactly the same feature, just at different locations in the input image
 - convolutional networks are well adapted to the translation invariance of images
- Example
 - 96 unique set of weights (one for each depth slice),
 - for a total of $96 * 11 * 11 * 3 = 34,848$ unique weights,
 - 34,944 parameters (+96 biases).
- if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a **convolution** of the neuron's weights with the input volume → the name: Convolutional Layer
- Set of weights = filter (kernel)
- Can be applied, but are image-dependent
 - faces that have been centered in the image
- A Convolutional Layer without parameter sharing → **Locally-Connected Layer**

Conv NNs

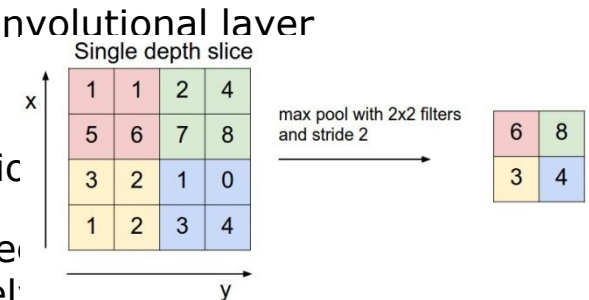
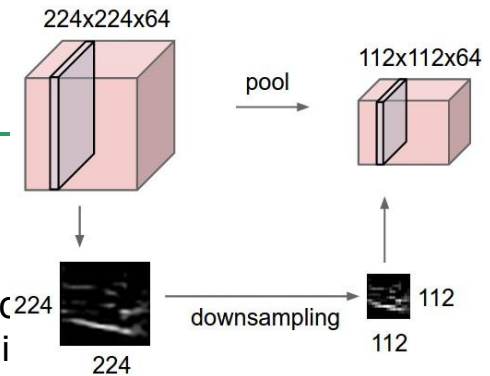
□ Pooling layer

■ Aim

- progressively reduce the spatial size of the representation
 - to reduce the amount of parameters and computation in the network
 - to also control overfitting
- downsample the spatial dimensions of the input.
- simplify the information in the output from the convolutional layer

■ How it works

- takes each feature map output from the convolutional layer and condenses it into a smaller feature map
- each unit in the pooling layer may summarize a region of the input feature map
- apply pooling filters to each feature map separately,
 - Pooling filter size (spatial extent of pooling) P_F
 - Pooling filter stride P_S
 - No padding
- resizes it spatially, using
 - the MAX operation
 - the average operation
 - L2-norm operation (square root of the sum of the squares of the activations in the 2×2 region)
 - Lp norm L_p : $\sqrt[p]{\sum (X^p)}$
 - Log prob PROB: $1/b \log(\sum(\exp(bX)))$



Conv NNs

□ Pooling layer

■ Size conversion

□ Input:

- $K \times N$

□ Output

- $K \times [(N - PF) / PS + 1]$

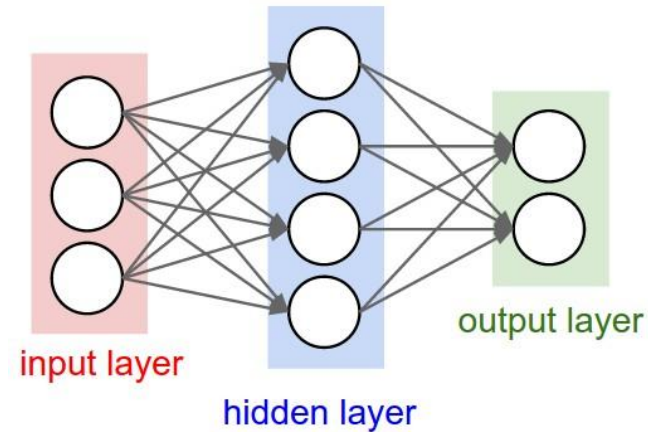
■ Remark

- introduces zero parameters since it computes a fixed function of the input
- note that it is not common to use zero-padding for Pooling layers
- pooling layer with $PF=3, PS=2$ (also called overlapping pooling), and more commonly $PF=2, PS=2$
- pooling sizes with larger filters are too destructive
- keep track of the index of the max activation (sometimes also called *the switches*) so that gradient routing is efficient during backpropagation

Conv NNs

□ Fully-connected layer

- Neurons have full connections to all inputs from the previous layer



- Various activations
 - ReLU (often)

Conv NNs

□ CNN architectures

- INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

- Most common:

- INPUT -> FC \leftrightarrow a linear classifier

- INPUT -> CONV -> RELU -> FC

- INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC.

- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC

- a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation

Conv NNs

□ Remarks

- Prefer a stack of small filter CONV to one large receptive field CONV layer
 - Pro:
 - Non-linear functions
 - Few parameters
 - Cons:
 - more memory to hold all the intermediate CONV layer results
- Input layer size → divisible by 2 many times
- Conv layers → small filters
 - $S \geq 1$
 - $P = (F - 1) / 2$
- Pool layers
 - $F \leq 3, S = 2$

Conv NNs

□ Output layer

■ Multiclass SVM

- Largest score indicates the correct answer

■ Softmax (normalized exponential function)

- Largest probability indicates the correct answer
- converts raw scores to probabilities
- "squashes" a $\#classes$ -dimensional vector \mathbf{z} of arbitrary real values to a $\#classes$ -dimensional vector $\sigma(\mathbf{z})$ of real values in the range $(0, 1)$ that add up to 1
- $\sigma(\mathbf{z})_j = \exp(z_j) / \sum_{k=1.. \#classes} \exp(z_k)$

Conv NNs

□ Common architectures

- LeNet (Yann LeCun, 1998) - <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>
 - A conv layer + a pool layer
- AlexNet (Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, 2012) <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
 - More conv layers + more pool layers
- ZF Net (Matthew Zeiler and Rob Fergus, 2013) <https://arxiv.org/pdf/1311.2901.pdf>
 - AlexNet + optimisation of hyper-parameters
- GoogleLeNet (Christian Szegedy et al., 2014) <https://arxiv.org/pdf/1409.4842.pdf>
 - *Inception Module* that dramatically reduced the number of parameters in the network (AlexNet 60M, GoogleLeNet 4M) <https://arxiv.org/pdf/1602.07261.pdf>
 - uses Average Pooling instead of Fully Connected layers at the top of the ConvNet → eliminating parameters
- VGGNet (Karen Simonyan and Andrew Zisserman, 2014) <https://arxiv.org/pdf/1409.1556.pdf>
 - 16 Conv/FC layers (FC → a lot more memory; they can be eliminated)
 - pretrained model is available for plug and play use in Caffe
- ResNet (Kaiming He et al., 2015) <https://arxiv.org/pdf/1512.03385.pdf> (Torch)
 - *skip connections*
 - batch normalization

Conv NNs

□ Reducing overfitting

■ increasing the amount of training data

- Artificially expanding the training data
 - Rotations, adding noise,

■ reduce the size of the network

- Not recommended

■ regularization techniques

□ Effect:

- the network prefers to learn small weights, all other things being equal. Large weights will only be allowed if they considerably improve the first part of the cost function
- a way of compromising between finding small weights and minimizing the original cost function (when λ is small we prefer to minimize the original cost function, but when λ is large we prefer small weights)
- Give importance to all features
 - $X = [1, 1, 1, 1]$
 - $W_1 = [1, 0, 0, 0]$
 - $W_2 = [0.25, 0.25, 0.25, 0.25]$
 - $W_1^T X = W_2^T X = 1$
 - $L_1(W_1) = 0.25 + 0.25 + 0.25 + 0.25 = 1$
 - $L_1(W_2) = 1 + 0 + 0 + 0 = 1$

Conv NNs

□ Reducing overfitting - regularization techniques

■ Methods

- L1 regularisation – add the sum of the absolute values of the weights $C = C_0 + \lambda/n \sum |w|$
 - the weights shrink by a constant amount toward 0
 - Sparsity (feature selection – more weights are 0)
- *weight decay (L2 regularization)* - add an extra term to the cost function (the L2 regularization term = the sum of the squares of all the weights in the network = $\lambda/2n \sum w^2$): $C = C_0 + \lambda/2n \sum w^2$
 - the weights shrink by an amount which is proportional to w
- Elastic net regularisation
 - $\lambda_1|w| + \lambda_2 w^2$
- Max norm constraints (clapping)
- Dropout - modify the network itself (<http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>)
 - Some neurons are temporarily deleted
 - propagate the input and backpropagate the result through the modified network
 - update the appropriate weights and biases.
 - repeat the process, first restoring the dropout neurons, then choosing a new random subset of hidden neurons to delete

Improve NN's performance

- Cost functions → loss functions
- Regularisation
- Initialisation of weights
- NN's hyper-parameters

Improve NN's performance

□ Cost functions → loss functions

■ Possible cost functions

□ Quadratic cost

- $1/2n \sum_x \|D - C\|^2$

□ Cross-entropy loss (negative log likelihood)

- $-1/n \sum_x [D \ln C + (1 - D) \ln(1 - C)]$

■ Optimizing the cost function

□ Stochastic gradient descent by backpropagation

□ Hessian technique

- Pro: it incorporates not just information about the gradient, but also information about how the gradient is changing
- Cons: the sheer size of the Hessian matrix

□ Momentum-based gradient descent

- Velocity & friction

Improve NN's performance

□ Initialisation of weights

■ Pitfall

- all zero initialization

■ Small random numbers

- $W = 0.01 * \text{random}(D, H)$

■ Calibrating the variances with $1/\sqrt{\#\text{Inputs}}$

- $w = \text{random}(\#\text{Inputs}) / \sqrt{\#\text{Inputs}}$

■ Sparse initialization

■ Initializing the biases

■ In practice

- $w = \text{random}(\#\text{Inputs}) * \sqrt{2.0/\#\text{Inputs}}$

Improve NN's performance

□ NN's hyper-parameters*

■ Learning rate η

- Constant rate

- Not-constant rate

■ Regularisation parameter λ

■ Mini-batch size

*see Bengio's papers: <https://arxiv.org/pdf/1206.5533v2.pdf> and <http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf> or Snock's paper <http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf>

Improve NN's performance

□ NN's hyper-parameters

■ Learning rate η

- Constant rate

- Not-constant rate

 - Annealing the learning rate

 - Second order methods

 - Per-parameter adaptive learning rate methods

Improve NN's performance

- NN's hyper-parameters - Learning rate η
 - Not-constant rate
 - Annealing the learning rate
 - **Step decay**
 - Reduce the learning rate by some factor every few epochs
 - $\eta = \eta * \text{factor}$
 - Eg. $\eta = \eta * 0.5$ every 5 epochs
 - Eg. $\eta = \eta * 0.1$ every 20 epochs
 - Exponential decay
 - $a = a_0 \exp(-kt)$,where a_0, k are hyperparameters and t is the iteration number (but you can also use units of epochs).
 - 1/t decay
 - $a = a_0 / (1 + kt)$where a_0, k are hyperparameters and t is the iteration number.

Improve NN's performance

- NN's hyper-parameters - Learning rate η
 - Not-constant rate
 - Second order methods
 - Newton's method (Hessian)
 - *quasi-Newton* methods
 - L- BGFS (Limited memory Broyden-Fletcher-Goldfarb-Shanno)
 - https://static.googleusercontent.com/media/research.google.com/ro//archive/large_deep_networks_nips2012.pdf
 - <https://arxiv.org/pdf/1311.2115.pdf>

Improve NN's performance

- NN's hyper-parameters - Learning rate η
 - Not-constant rate
 - Per-parameter adaptive learning rate methods
 - Adagrad
 - <http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>
 - RMSprop
 - http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
 - Adam
 - <https://arxiv.org/pdf/1412.6980.pdf>

Deep NNs

□ Typology

- <http://www.asimovinstitute.org/neural-network-zoo/>
- <https://uvadlc.github.io/lectures/lecture1.pdf>

- Feedforward NNs
 - MLPs

- CNNs
 - Inception
 - Residual nets

- Recurrent NNs
 - LSTM

- Recursive NNs

Tools

- Keras
 - NN API
 - <https://keras.io/>
 - + Theano (machine learning library; multi-dim arrays)
<http://www.deeplearning.net/software/theano/>
http://www.iro.umontreal.ca/~lisa/pointeurs/theano_scipy2010.pdf
 - + TensorFlow (numerical computation) <https://www.tensorflow.org/>
- Pylearn2 <http://deeplearning.net/software/pylearn2/>
 - ML library
 - + Theano
- Torch <http://torch.ch/>
 - scientific computing framework
 - Multi-dim array
 - NN
 - GPU
- Caffe
 - deep learning framework
 - Berkley

Învățare supervizată – algoritmi

Mașini cu suport vectorial (MSV)

- Dezvoltate de Vapnik în 1970
- Popularizate după 1992
- Clasificatori liniari care identifică un hiperplan de separare între clasa pozitivă și cea negativă
- Au o fundamentare teoretică foarte riguroasă
- Funcționează foarte bine pentru date de volum mare
 - analiza textelor,
 - analiza imaginilor

Învățare supervizată – algoritmi

Mașini cu suport vectorial

Concepte de bază

□ Un set de date

■ De antrenament

- $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$, unde
 - $\mathbf{x}_i = (x_1, x_2, \dots, x_m)$ este un vector de intrare într-un spațiu real $X \subseteq R^m$ și
 - y_i este eticheta clasei (valoarea de ieșire), $y_i \in \{1, -1\}$
 - $1 \rightarrow$ clasă pozitivă, $\langle \mathbf{w}, \mathbf{x}_i \rangle + b \geq 0$
 - $-1 \rightarrow$ clasă negativă, $\langle \mathbf{w}, \mathbf{x}_i \rangle + b < 0$

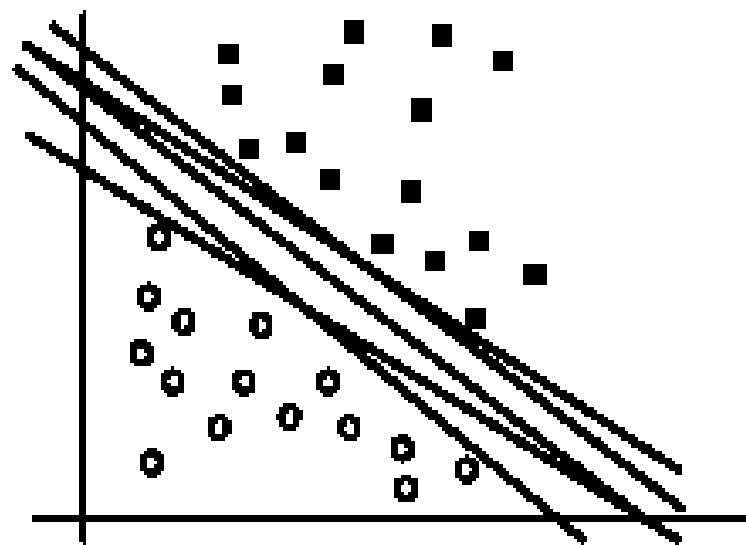
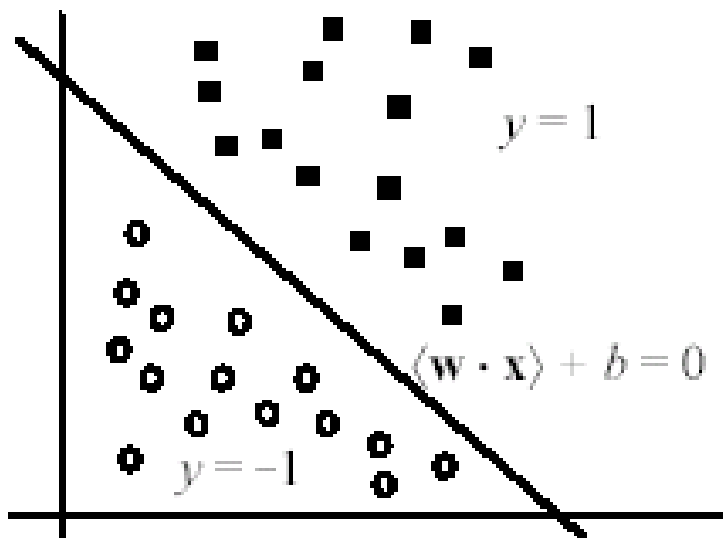
□ MSV găsește o funcție liniară de forma

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b, (\mathbf{w}: \text{vector pondere})$$

Învățare supervizată – algoritmi

Mașini cu suport vectorial

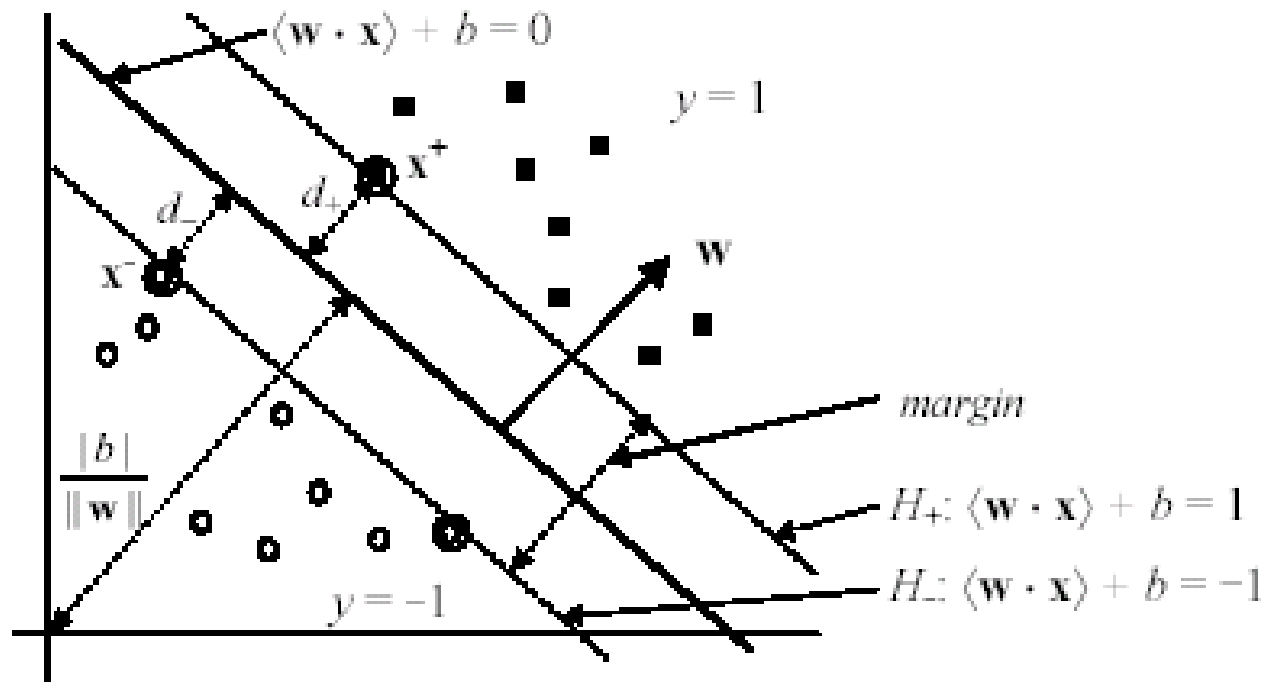
- Hiperplanul de decizie care separă cele 2 clase este:
 - $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$
- Pot exista mai multe hiperplanuri
 - Care este cel mai bun?



Învățare supervizată – algoritmi

Mașini cu suport vectorial

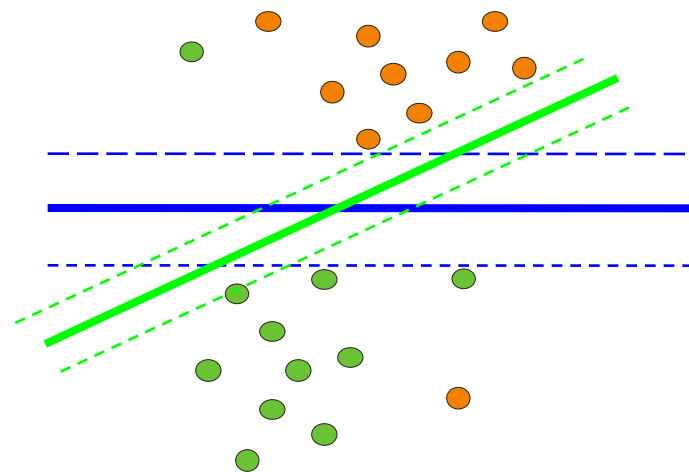
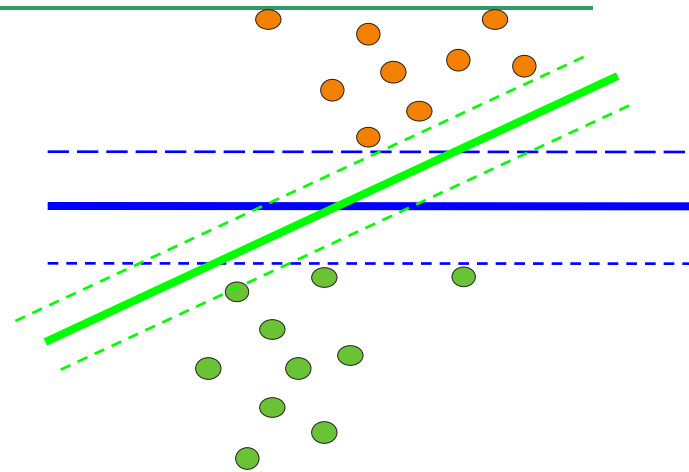
- MSV caută hiperplanul cu cea mai largă margine (cel care micșorează eroarea de generalizare)
 - Algoritmul SMO (*Sequential minimal optimization*)



Învățare supervizată – algoritmi

Mașini cu suport vectorial

- Cazuri de date
 - Liniar separabile
 - Separabile
 - Eroarea = 0
 - Ne-separabile
 - Se relaxează constrângerile → se permit unele erori
 - C – coeficient de penalizare



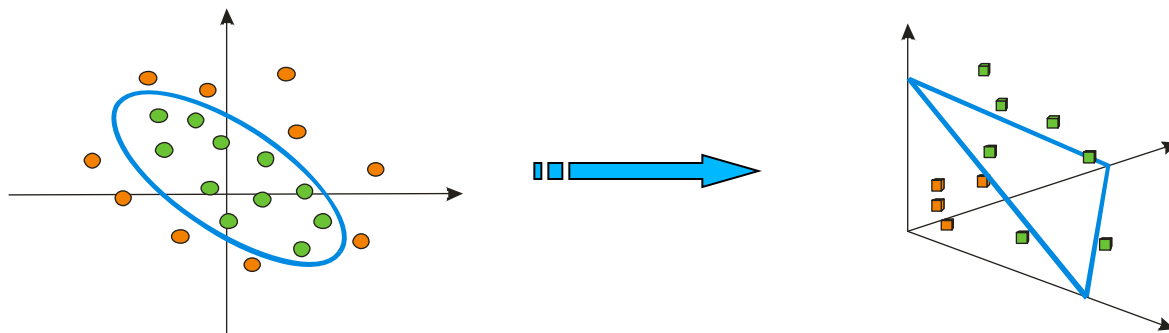
Învățare supervizată – algoritmi

Mașini cu suport vectorial

Cazuri de date

□ Non-linear separabile

- Spațiul de intrare se transformă într-un spațiu cu mai multe dimensiuni (*feature space*), cu ajutorul unei funcții kernel, unde datele devin linear separabile



■ Kernele posibile

□ Clasice

- Polynomial kernel: $K(\mathbf{x}_1, \mathbf{x}_2) = (\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1)^d$
- RBF kernel: $K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\sigma \|\mathbf{x}_1 - \mathbf{x}_2\|^2)$

□ Kernele multiple

- Liniare: $K(\mathbf{x}_1, \mathbf{x}_2) = \sum w_i K_i$
- Ne-liniare
 - Fără coeficienți: $K(\mathbf{x}_1, \mathbf{x}_2) = K_1 + K_2 * \exp(K_3)$
 - Cu coeficienți: $K(\mathbf{x}_1, \mathbf{x}_2) = K_1 + c_1 * K_2 * \exp(c_2 + K_3)$

Învățare supervizată – algoritmi

Mașini cu suport vectorial

□ Probleme

- Doar attribute reale
- Doar clasificare binară
- Background matematic dificil

□ Tool-uri

- LibSVM → <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- Weka → SMO
- SVMLight → <http://svmlight.joachims.org/>
- SVM Torch → <http://www.torch.ch/>
- <http://www.support-vector-machines.org/>

Învățare supervizată – algoritmi

Regresie

- Studiul legăturii între variabile
- Se dă
 - un set de date (exemple, instanțe, cazuri)
 - date de antrenament – sub forma unor perechi ($attribute_data_i, ieșire_i$), unde
 - $i = 1, N$ ($N =$ nr datelor de antrenament)
 - **$attribute_data_i = (atr_{i1}, atr_{i2}, \dots, atr_{im})$** , m – nr atributelor (caracteristicilor, proprietăților) unei date
 - **$ieșire_i$ – un număr real**
 - date de test
 - sub forma (**$attribute_data_i$**), $i = 1, n$ ($n =$ nr datelor de test)
- Să se determine
 - o funcție (necunoscută) care realizează corespondența attribute – ieșire pe datele de antrenament
 - Ieșirea (valoarea) asociată unei date (noi) de test folosind funcția învățată pe datele de antrenament
- Cum găsim forma (expresia) funcției?
 - Algoritmi evolutivi → Programare genetică

Învățare supervizată – algoritmi

Algoritmi evolutivi

Algoritmi

- Inspirați din natură (biologie)
- Iterativi
- Bazați pe
 - populații de potențiale soluții
 - căutare aleatoare ghidată de
 - Operații de selecție naturală
 - Operații de încrucișare și mutație
- Care procesează în paralel mai multe soluții

Metafora evolutivă

Evoluție naturală	Rezolvarea problemelor
Individ	Soluție potențială (candidat)
Populație	Mulțime de soluții
Cromozom	Codarea (reprezentarea) unei soluții
Genă	Parte a reprezentării
Fitness (măsură de adaptare)	Calitate
Încrucișare și mutație	Operatori de căutare
Mediu	Spațiul de căutare al problemei

Învățare supervizată – algoritmi

Algoritmi evolutivi

Initializare populație $P(0)$

Evaluare $P(0)$

$g := 0$; //generația

CâtTimp (not condiție_stop) execută

Repetă

Selectează 2 părinți $p1$ și $p2$ din $P(g)$

Încrucișare($p1, p2$) \Rightarrow $o1$ și $o2$

Mutație($o1$) \Rightarrow $o1^*$

Mutație($o2$) \Rightarrow $o2^*$

Evaluare($o1^*$)

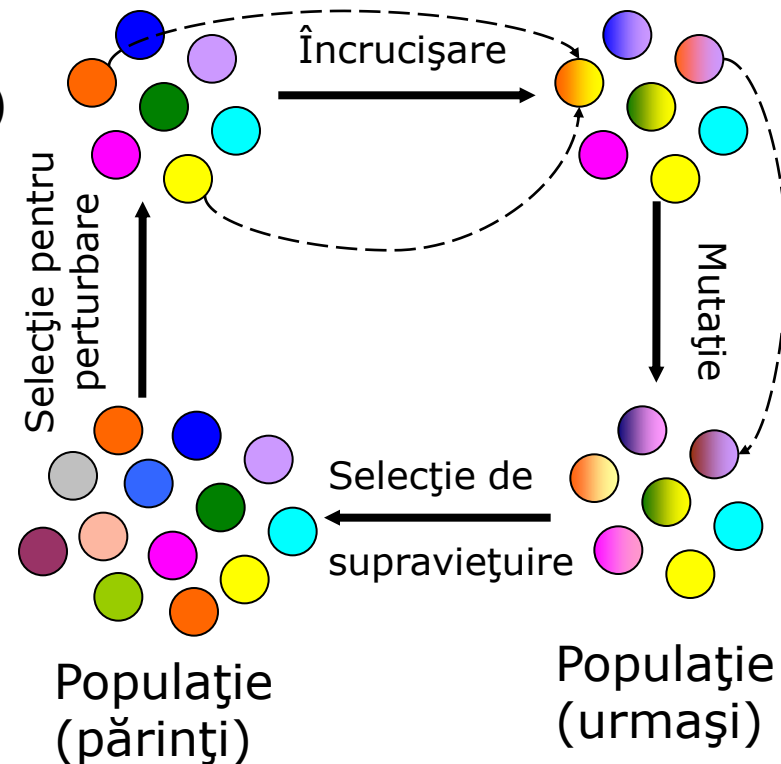
Evaluare($o2^*$)

adăugare $o1^*$ și $o2^*$ în $P(g+1)$

Până când $P(g+1)$ este completă

$g := g + 1$

Sf CâtTimp



Învățare supervizată – algoritmi

Algoritmi evolutivi → Programare genetică

- Un tip particular de algoritmi evolutivi
- Cromozomi
 - sub formă de arbore care codează mici programe
- Fitness-ul unui cromozom
 - Performanța programului codat în el

- <http://www.genetic-programming.org/>

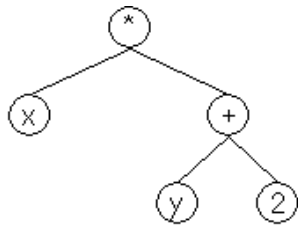
Învățare supervizată – algoritmi

Algoritmi evolutivi → Programare genetică

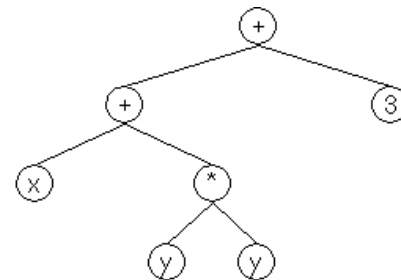
□ Reprezentare

- Cromozomul = un arbore cu noduri de tip
 - Funcție → operatori matematici ($+, -, *, /, \sin, \log, \dots$)
 - Terminal → attribute ale datelor problemei sau constante (x, y, z, a, b, c, \dots)
- care codează expresia matematică a unei funcții

$$x(y+2)$$



$$x+y^2+3$$



Învățare supervizată – algoritmi

Algoritmi evolutivi → Programare genetică

Fitness

- Eroarea de predicție
- pp următoarele date de intrare (2 atribute și o ieșire) și 2 cromozomi:

- $c_1 = 3x_1 - x_2 + 5$

- $c_2 = 3x_1 + 2x_2 + 2$

$f^*(x_1, x_2) = 3x_1 + 2x_2 + 1$ – necunoscută

x_1	x_2	$f^*(x_1, x_2)$	$f_1(x_1, x_2)$	$f_2(x_1, x_2)$	$ f^* - f_1 $	$ f^* - f_2 $
1	1	6	7	7	1	1
0	1	3	4	4	1	1
1	0	4	8	5	4	1
-1	1	0	1	1	1	1
					$\Sigma = 7$	$\Sigma = 4$

→ c_2 e mai bun
ca c_1

Învățare supervizată – algoritmi

Algoritmi evolutivi → Programare genetică

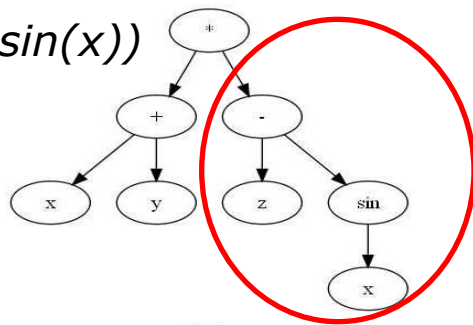
□ Inițializare

- Generare aleatoare de arbori corecți → expresii matematice valide

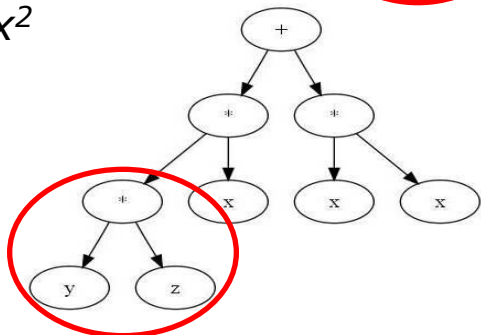
□ Încrucișare

- Cu punct de tăietură – se interchimbă doi sub-arbori

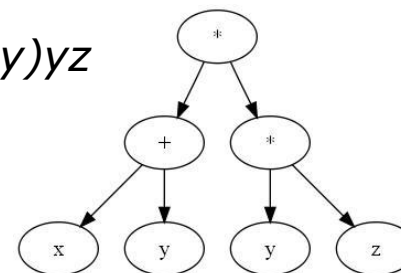
$$p_1 = (x+y) * (z - \sin(x))$$



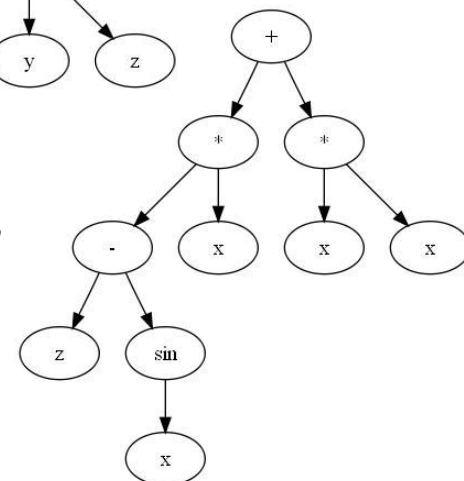
$$p_2 = xyz + x^2$$



$$f_1 = (x+y)yz$$



$$f_2 = (z - \sin(x))x + x^2$$



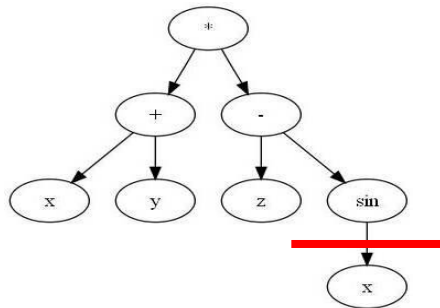
Învățare supervizată – algoritmi

Algoritmi evolutivi → Programare genetică

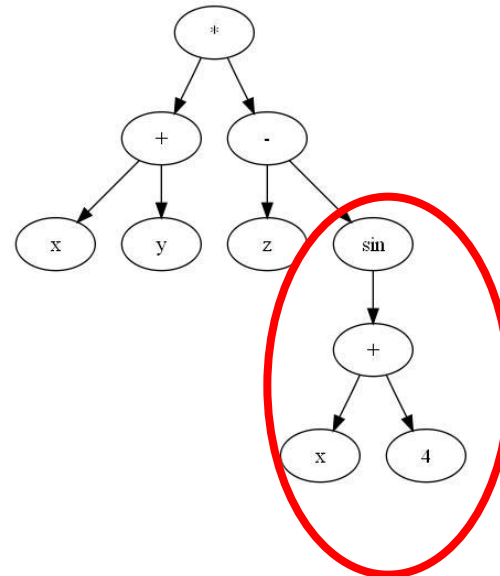
□ Mutație

■ Generarea unui nou sub-arbore

$$p = (x + y) * (z - \sin(x))$$



$$f = (x + y) * (z - \sin(x + 4))$$



Învățare automată

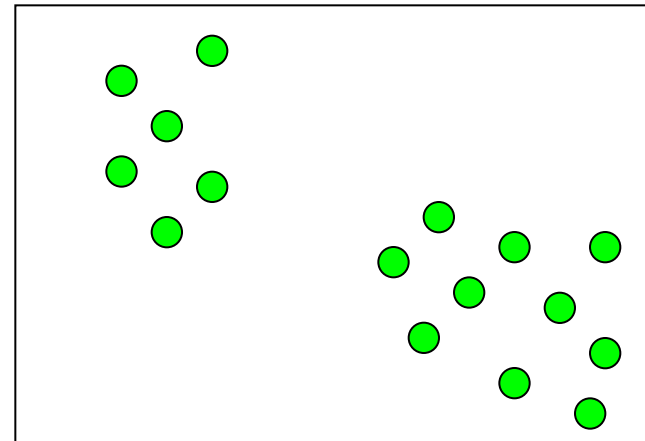
- Învățare supervizată
- **Învățare ne-supervizată**
- Învățare cu întărire
- Teoria învățării

Învățare nesupervizată

- Scop
 - Găsirea unui model sau a unei structuri utile a datelor

- Tip de probleme
 - Identificarea unor grupuri (clusteri)
 - Analiza genelor
 - Procesarea imaginilor
 - Analiza rețelelor sociale
 - Segmentarea pieței
 - Analiza datelor astronomice
 - Clusteri de calculatoare
 - Reducerea dimensiunii
 - Identificarea unor cauze (explicații) ale datelor
 - Modelarea densității datelor

- Caracteristic
 - Datele nu sunt adnotate (etichetate)



Învățare ne-supervizată – definiere

Împărțirea unor exemple **neetichetate** în submulțimi disjuncte (clusteri) astfel încât:

- exemplele din același cluster sunt foarte similare
- exemplele din clusteri diferiți sunt foarte diferite

Definire

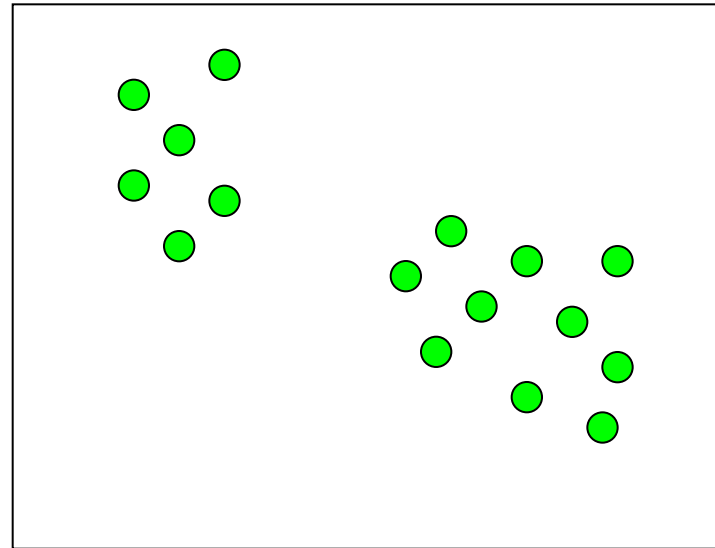
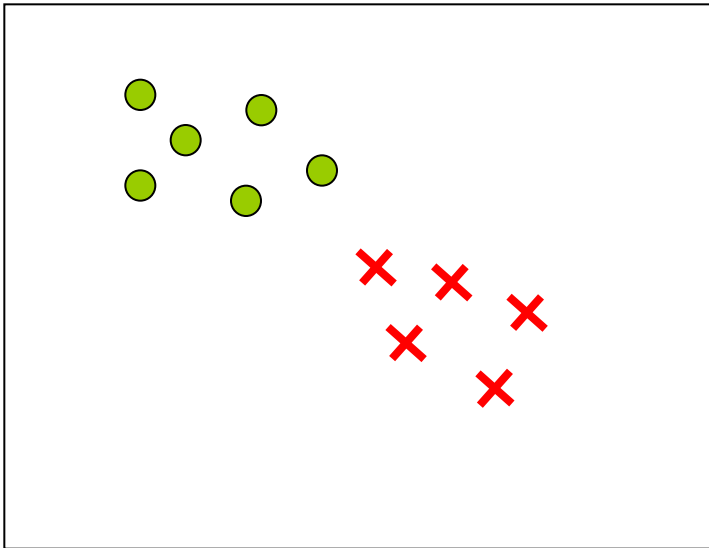
- Se dă
 - un set de date (exemple, instanțe, cazuri)
 - Date de antrenament
 - Sub forma **attribute_data_i**, unde
 - $i = 1, N$ ($N =$ nr datelor de antrenament)
 - **attribute_data_i** = $(atr_{i1}, atr_{i2}, \dots, atr_{im})$, m – nr atributelor (caracteristicilor, proprietăților) unei date
 - Date de test
 - Sub forma (**attribute_data_i**), $i = 1, n$ ($n =$ nr datelor de test)
 - Se determină
 - o funcție (necunoscută) care realizează gruparea datelor de antrenament în mai multe clase
 - Nr de clase poate fi pre-definit (k) sau necunoscut
 - Datele dintr-o clasă sunt asemănătoare
 - clasa asociată unei date (noi) de test folosind gruparea învățată pe datele de antrenament

Alte denumiri

- Clustering

Învățare ne-supervizată – definiere

□ Supervizată vs. Ne-supervizată



Învățare ne-supervizată – definiere

- Distanțe între 2 elemente \mathbf{p} și $\mathbf{q} \in R^m$
 - Euclideană
 - $d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{j=1,2,\dots,m} (p_j - q_j)^2}$
 - Manhattan
 - $d(\mathbf{p}, \mathbf{q}) = \sum_{j=1,2,\dots,m} |p_j - q_j|$
 - Mahalanobis
 - $d(\mathbf{p}, \mathbf{q}) = \sqrt{(\mathbf{p} - \mathbf{q})^T S^{-1} (\mathbf{p} - \mathbf{q})}$,
 - unde S este matricea de variație și covariație ($S = E[(\mathbf{p} - E[\mathbf{p}])(\mathbf{q} - E[\mathbf{q}])^T]$)
 - Produsul intern
 - $d(\mathbf{p}, \mathbf{q}) = \sum_{j=1,2,\dots,m} p_j q_j$
 - Cosine
 - $d(\mathbf{p}, \mathbf{q}) = \sum_{j=1,2,\dots,m} p_j q_j / (\sqrt{\sum_{j=1,2,\dots,m} p_j^2} * \sqrt{\sum_{j=1,2,\dots,m} q_j^2})$
 - Hamming
 - numărul de diferențe între \mathbf{p} și \mathbf{q}
 - Levenshtein
 - numărul minim de operații necesare pentru a-l transforma pe \mathbf{p} în \mathbf{q}

- Distanță vs. Similaritate
 - Distanța \rightarrow min
 - Similaritatea \rightarrow max

Învățare ne-supervizată – exemple

- Gruparea genelor
- Studii de piață pentru gruparea clienților (segmentarea pieței)
- news.google.com

Învățare ne-supervizată – proces

Procesul

- 2 pași:
 - Antrenarea
 - Învățarea (determinarea), cu ajutorul unui algoritm, a clusterilor existenți
 - Testarea
 - Plasarea unei noi date într-unul din clusterii identificați în etapa de antrenament

Calitatea învățării (validarea clusterizării):

- Criterii interne
 - Similaritate ridicată în interiorul unui cluster și similaritate redusă între clusteri
- Criterii externe
 - Folosirea unor benchmark-uri formate din date pre-grupate

Învățare ne-supervizată – evaluare

Măsuri de performanță

□ Criterii interne

- Distanța în interiorul clusterului
- Distanța între clusteri
- Indexul Davies-Bouldin
- Indexul Dunn

□ Criterii externe

- Compararea cu date cunoscute – în practică este imposibil
- Precizia
- Rapelul
- F-measure

Învățare ne-supervizată – evaluare

Măsuri de performanță

□ Criterii interne

- Distanța în interiorul clusterului c_j care conține n_j instanțe
 - Distanța medie între instanțe (average distance)
 - $D_a(c_j) = \sum_{x_{i1}, x_{i2} \in c_j} \|x_{i1} - x_{i2}\| / (n_j(n_j-1))$
 - Distanța între cei mai apropiați vecini (nearest neighbour distance)
 - $D_{nn}(c_j) = \sum_{x_{i1} \in c_j} \min_{x_{i2} \in c_j} \|x_{i1} - x_{i2}\| / n_j$
 - Distanța între centroizi
 - $D_c(c_j) = \sum_{x_i \in c_j} \|x_i - \mu_j\| / n_j$, unde $\mu_j = 1/n_j \sum_{x_i \in c_j} x_i$

Învățare ne-supervizată – evaluare

Măsuri de performanță

□ Criterii interne

■ Distanța între 2 clusteri c_{j1} și c_{j2}

□ Legătură simplă

$$d_s(c_{j1}, c_{j2}) = \min_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{ \|x_{i1} - x_{i2}\| \}$$

□ Legătură completă

$$d_{co}(c_{j1}, c_{j2}) = \max_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{ \|x_{i1} - x_{i2}\| \}$$

□ Legătură medie

$$d_a(c_{j1}, c_{j2}) = \sum_{x_{i1} \in c_{j1}, x_{i2} \in c_{j2}} \{ \|x_{i1} - x_{i2}\| \} / (n_{j1} * n_{j2})$$

□ Legătură între centroizi

$$d_{ce}(c_{j1}, c_{j2}) = \| \mu_{j1} - \mu_{j2} \|$$

Învățare ne-supervizată – evaluare

Măsuri de performanță

□ Criterii interne

■ Indexul Davies-Bouldin → min → clusteri compacți

- $DB = 1/nc * \sum_{i=1,2,\dots,nc} \max_{j=1, 2, \dots, nc, j \neq i} ((\sigma_i + \sigma_j)/d(\mu_i, \mu_j))$

- unde:

- nc – numărul de clusteri
- μ_i – centroidul clusterului i
- σ_i – media distanțelor între elementele din clusterul i și centroidul μ_i
- $d(\mu_i, \mu_j)$ – distanța între centroidul μ_i și centroidul μ_j

■ Indexul Dunn

- Identifică clusterii denși și bine separați

- $D = d_{min}/d_{max}$

- Unde:

- d_{min} – distanța minimă între 2 obiecte din clusteri diferiți – distanța intra-cluster
- d_{max} – distanța maximă între 2 obiecte din același cluster – distanța inter-cluster

Învățare ne-supervizată - tipologie

- După modul de formare al clusterilor
 - C. ierarhic
 - C. ne-ierarhic (partițional)
 - C. bazat pe densitatea datelor
 - C. bazat pe un grid

Învățare ne-supervizată - tipologie

- După modul de formare al clusterilor
 - Ierarhic
 - se crează un arbore taxonomic (dendogramă)
 - crearea clusterilor (recursiv)
 - nu se cunoaște k (nr de clusteri)
 - aglomerativ (de jos în sus) → clusteri mici spre clusteri mari
 - diviziv (de sus în jos) → clusteri mari spre clusteri mici
 - Ex. Clustering ierarhic aglomerativ

Învățare ne-supervizată - tipologie

- După modul de formare al clusterilor
 - Ne-ierarhic
 - Partițional → se determină o împărțire a datelor → toți clusterii deodată
 - Optimizează o funcție obiectiv definită
 - Local – doar pe anumite atribute
 - Global – pe toate atributelecare poate fi
 - Pătratul erorii – suma patratelor distanțelor între date și centroizii clusterilor → min
 - Ex. *K-means*
 - Bazată pe grafuri
 - Ex. Clusterizare bazată pe arborele minim de acoperire
 - Bazată pe modele probabilistice
 - Ex. Identificarea distribuției datelor → Maximizarea așteptărilor
 - Bazată pe cel mai apropiat vecin
 - Necesită fixarea *a priori* a lui k → fixarea clusterilor inițiali
 - Algoritmii se rulează de mai multe ori cu diferiți parametri și se alege versiunea cea mai eficientă
 - Ex. *K-means*, *ACO*

Învățare ne-supervizată - tipologie

- După modul de formare al clusterilor
 - bazat pe densitatea datelor
 - Densitatea și conectivitatea datelor
 - Formarea clusterilor de bazează pe densitatea datelor într-o anumită regiune
 - Formarea clusterilor de bazează pe conectivitatea datelor dintr-o anumită regiune
 - Funcția de densitate a datelor
 - Se încearcă modelarea legii de distribuție a datelor
 - Avantaj:
 - Modelarea unor clusteri de orice formă

Învățare ne-supervizată - tipologie

- După modul de formare al clusterilor
 - Bazat pe un grid
 - Nu e chiar o metodă nouă de lucru
 - Poate fi ierarhic, partițional sau bazat pe densitate
 - Pp. segmentarea spațiului de date în zone regulate
 - Obiectele se plasează pe un grid multi-dimensional
 - Ex. ACO

Învățare ne-supervizată - tipologie

- După modul de lucru al algoritmului
 - Aglomerativ
 1. Fiecare instanță formează inițial un cluster
 2. Se calculează distanțele între oricare 2 clusteri
 3. Se reunesc cei mai apropiați 2 clusteri
 4. Se repetă pașii 2 și 3 până se ajunge la un singur cluster sau la un alt criteriu de stop
 - Diviziv
 1. Se stabilește numărul de clusteri (k)
 2. Se inițializează centrul fiecărui cluster
 3. Se determină o împărțire a datelor
 4. Se recalculază centrul clusterelor
 5. Se repetă pașii 3 și 4 până partiționarea nu se mai schimbă (algoritmul a converș)

- După atributele considerate
 - Monotetic – atributele se consideră pe rând
 - Politetic – atributele se consideră simultan

Învățare ne-supervizată - tipologie

- După tipul de apartenență al datelor la clusteri
 - Clustering exact (*hard clustering*)
 - Asociază fiecărei intrări \mathbf{x}_i o etichetă (clasă) c_j
 - Clustering fuzzy
 - Asociază fiecărei intrări \mathbf{x}_i un grad (probabilitate) de apartenență f_{ij} la o anumită clasă $c_j \rightarrow$ o instanță \mathbf{x}_i poate aparține mai multor clusteri

Învățare ne-supervizată – algoritmi

- Clustering ierarhic aglomerativ
- K-means
- AMA
- Modele probabilistice
- Cel mai apropiat vecin
- Fuzzy
- Rețele neuronale artificiale
- Algoritmi evolutivi
- ACO

Învățare ne-supervizată – algoritmi

Clustering ierarhic aglomerativ

- Se consideră o distanță între 2 instanțe $d(x_{i1}, x_{i2})$
- Se formează N clusteri, fiecare conținând câte o instanță
- Se repetă
 - Determinarea celor mai apropiați 2 clusteri
 - Se reunesc cei 2 clusteri → un singur cluster
- Până când se ajunge la un singur cluster (care conține toate instanțele)

Învățare ne-supervizată – algoritmi

Clustering ierarhic aglomerativ

- Distanța între 2 clusteri c_i și c_j :
 - Legătură simplă → minimul distanței între obiectele din cei 2 clusteri
 - $d(c_i, c_j) = \max_{x_{i1} \in c_i, x_{i2} \in c_j} \text{sim}(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
 - Legătură completă → maximul distanței între obiectele din cei 2 clusteri
 - $d(c_i, c_j) = \min_{x_{i1} \in c_i, x_{i2} \in c_j} \text{sim}(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
 - Legătură medie → media distanței între obiectele din cei 2 clusteri
 - $d(c_i, c_j) = 1 / (n_i * n_j) \sum_{x_{i1} \in c_i} \sum_{x_{i2} \in c_j} d(\mathbf{x}_{i1}, \mathbf{x}_{i2})$
 - Legătură medie peste grup → distanța între mediile (centrozii) celor 2 clusteri
 - $d(c_i, c_j) = \rho(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j)$, ρ – distanță, $\boldsymbol{\mu}_j = 1/n_j \sum_{x_{i2} \in c_j} \mathbf{x}_{i2}$

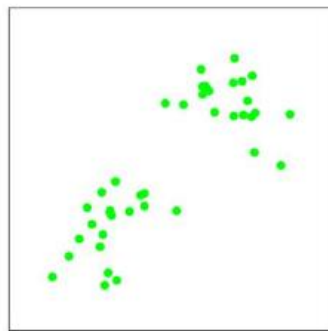
Învățare ne-supervizată – algoritmi

K-means (algoritmul Lloyd/iterația Voronoi)

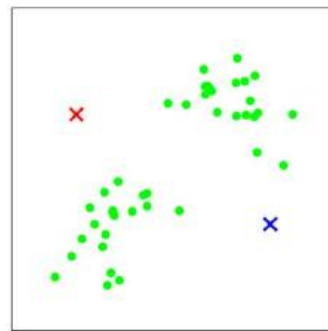
- Pp că se vor forma k clusteri
- Inițializează k centroizi $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k$
 - Un centroid $\boldsymbol{\mu}_j$ ($i=1, 2, \dots, k$) este un vector cu m valori (m – nr de attribute)
- Repetă până la convergență
 - Asociază fiecare instanță celui mai apropiat centroid → pentru fiecare instanță \mathbf{x}_i , $i = 1, 2, \dots, N$
 - $c_i = \arg \min_{j=1, 2, \dots, k} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$
 - Recalculează centroizii prin mutarea lor în media instanțelor asociate fiecăruia → pentru fiecare cluster c_j , $j = 1, 2, \dots, k$
 - $\boldsymbol{\mu}_j = \sum_{i=1, 2, \dots, N} \mathbf{1}_{c_i=j} \mathbf{x}_i / \sum_{i=1, 2, \dots, N} \mathbf{1}_{c_i=j}$

Învățare ne-supervizată – algoritmi

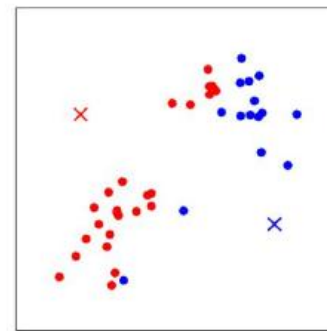
K-means



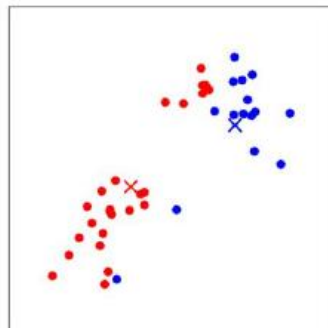
(a)



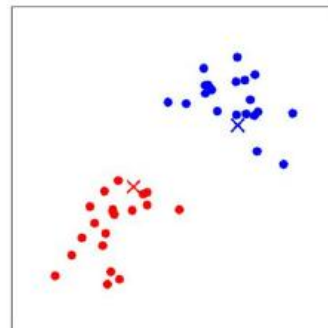
(b)



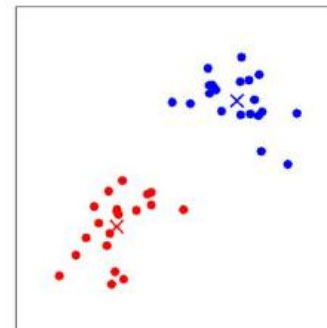
(c)



(d)



(e)



(f)

Învățare ne-supervizată – algoritmi

K-means

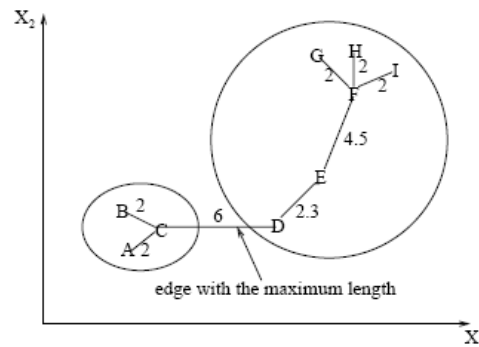
- Inițializarea a k centroizi $\mu_1, \mu_2, \dots, \mu_k$
 - Cu valori generate aleator (în domeniul de definiție al problemei)
 - Cu k dintre cele N instanțe (alese în mod aleator)

- Algoritmul converge întotdeauna?
 - Da, pt că avem funcția de distorsiune J
 - $J(c, \mu) = \sum_{i=1,2, \dots, N} \|\mathbf{x}_i - \mu_{c_j}\|^2$
care este descrescătoare
 - Converge într-un optim local
 - Găsirea optimului global \rightarrow NP-dificilă

Învățare ne-supervizată – algoritmi

Clusterizare bazată pe arborele minim de acoperire (AMA)

- Se construiește AMA al datelor
- Se elimină din arbore cele mai lungi muchii, formându-se clusteri



Învățare ne-supervizată – algoritmi

Modele probabilistice

- <http://www.gatsby.ucl.ac.uk/~zoubin/course04/ul.pdf>
- <http://learning.eng.cam.ac.uk/zoubin/nipstut.pdf>

Învățare ne-supervizată – algoritmi

Cel mai apropiat vecin

- Se etichetează câteva dintre instanțe
- Se repetă până la etichetarea tuturor instanțelor
 - O instanță ne-etichetată va fi inclusă în clusterul instanței cele mai apropiate
 - dacă distanța între instanța neetichetată și cea etichetată este mai mică decât un prag

Învățare ne-supervizată – algoritmi

Clusterizare fuzzy

- Se stabilește o partiționare fuzzy inițială
 - Se construiește matricea gradelor de apartenență U , unde u_{ij} – gradul de apartenență al instanței \mathbf{x}_i ($i=1,2, \dots, N$) la clusterul c_j ($j = 1, 2, \dots, k$) ($u_{ij} \in [0,1]$)
 - Cu cât u_{ij} e mai mare, cu atât e mai mare încrederea că instanța \mathbf{x}_i face parte din clusterul c_j

- Se stabilește o funcție obiectiv
 - $E^2(U) = \sum_{i=1,2, \dots, N} \sum_{j=1,2, \dots, k} u_{ij} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$,
 - unde $\boldsymbol{\mu}_j = \sum_{i=1,2, \dots, N} u_{ij} \mathbf{x}_i$ – centrul celui de-al j -lea fuzzy cluster
 - care se optimizează (min) prin re-atribuirea instanțelor (în clusteri noi)

- Clustering fuzzy \rightarrow clusterizare *hard* (fixă)
 - impunerea unui prag funcției de apartenență u_{ij}

Învățare ne-supervizată – algoritmi

Algoritmi evolutivi

Algoritmi

- Inspirați din natură (biologie)
- Iterativi
- Bazați pe
 - populații de potențiale soluții
 - căutare aleatoare ghidată de
 - Operații de selecție naturală
 - Operații de încrucișare și mutație
- Care procesează în paralel mai multe soluții

Metafora evolutivă

Evoluție naturală	Rezolvarea problemelor
Individ	Soluție potențială (candidat)
Populație	Mulțime de soluții
Cromozom	Codarea (reprezentarea) unei soluții
Genă	Parte a reprezentării
Fitness (măsură de adaptare)	Calitate
Încrucișare și mutație	Operatori de căutare
Mediu	Spațiul de căutare al problemei

Învățare ne-supervizată – algoritmi

Algoritmi evolutivi

Initializare populație $P(0)$

Evaluare $P(0)$

$g := 0$; //generația

CâtTimp (not condiție_stop) execută

Repetă

Selectează 2 părinți $p1$ și $p2$ din $P(g)$

Încrucișare($p1, p2$) \Rightarrow $o1$ și $o2$

Mutație($o1$) \Rightarrow $o1^*$

Mutație($o2$) \Rightarrow $o2^*$

Evaluare($o1^*$)

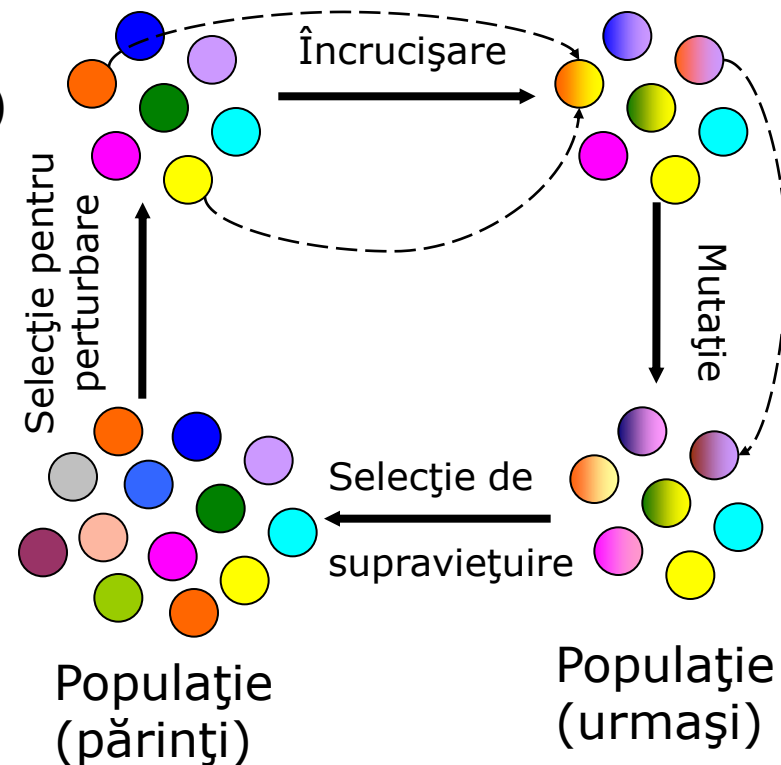
Evaluare($o2^*$)

adăugare $o1^*$ și $o2^*$ în $P(g+1)$

Până când $P(g+1)$ este completă

$g := g + 1$

Sf CâtTimp



Învățare ne-supervizată – algoritmi

Algoritmi evolutivi

- Reprezentare
 - Cromozomul = o partiționare a datelor
 - Ex. 2 clusteri → cromozom = vector binar
 - Ex. K clusteri → cromozom = vector cu valori din $\{1, 2, \dots, k\}$

- Fitness
 - Calitatea partiționării

- Inițializare
 - Aleatoare

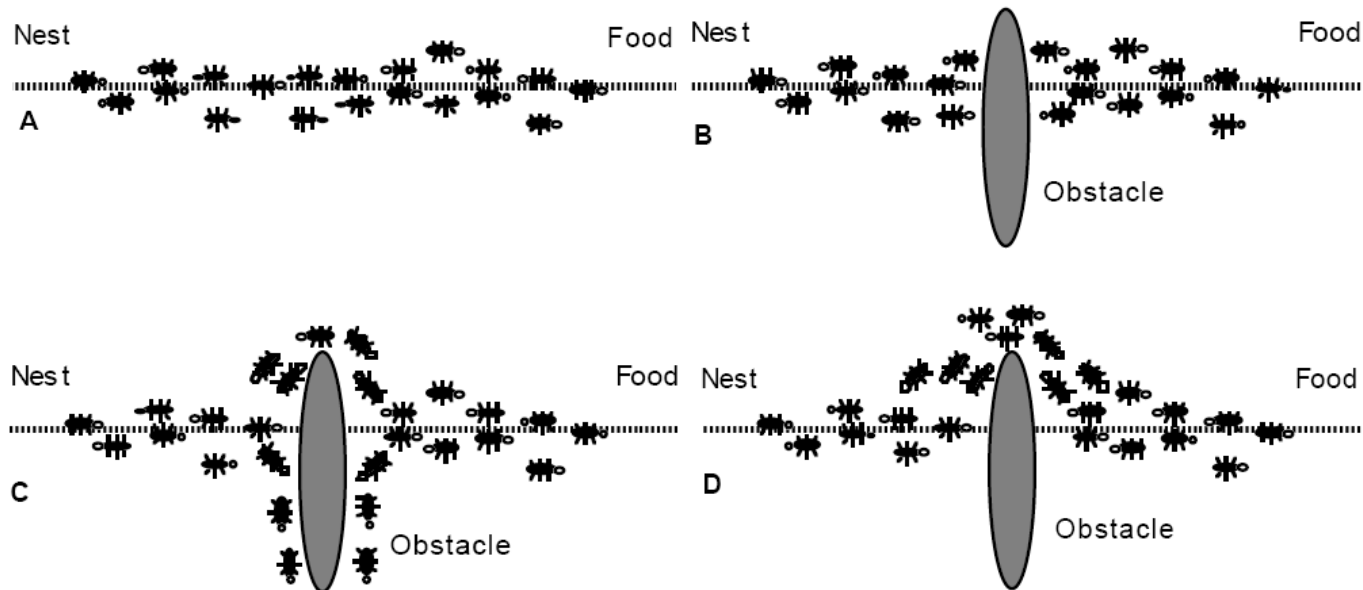
- Încrucișare
 - Punct de tăietură

- Mutație
 - Schimbarea unui element din cromozom

Învățare ne-supervizată – algoritmi

ACO

- ❑ Preferința pentru drumuri cu nivel ridicat de feromon
- ❑ Pe drumurile scurte feromonul se înmulțește
- ❑ Furnicile comunică pe baza urmelor de feromon



Învățare ne-supervizată – algoritmi

ACO

- ❑ Algoritm de clusterizare bazat pe un grid
- ❑ Obiectele se plasează aleator pe acest grid, urmând ca furnicuțele să le grupeze în funcție de asemănarea lor
- ❑ 2 reguli pentru furnicuțe
 - Furnica "ridică" un obiect-obstacol
 - ❑ Probabilitatea de a-l ridica e cu atât mai mare cu cât obiectul este mai izolat (în apropierea lui nu se află obiecte similare)
 - ❑ $p(\text{ridica}) = (k^+ / (k^+ + f))^2$
 - Furnica "depune" un obiect (anterior ridicat) într-o locație nouă
 - ❑ Probabilitatea de a-l depune e cu atât mai mare cu cât în vecinătatea locului de plasare se afla mai multe obiecte asemănătoare
 - ❑ $p(\text{depune}) = (f / (k^- + f))^2$
 - k^+ , k^- - constante
 - f – procentul de obiecte similare cu obiectul curent din memoria furnicuței
- ❑ Furnicuțele
 - au memorie
 - ❑ rețin obiectele din vecinătatea poziției curente
 - se mișcă ortogonal (N, S, E, V) pe grid pe căsuțele neocupate de alte furnici

Învățare automată

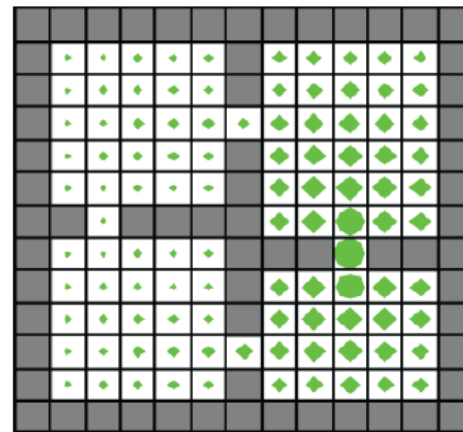
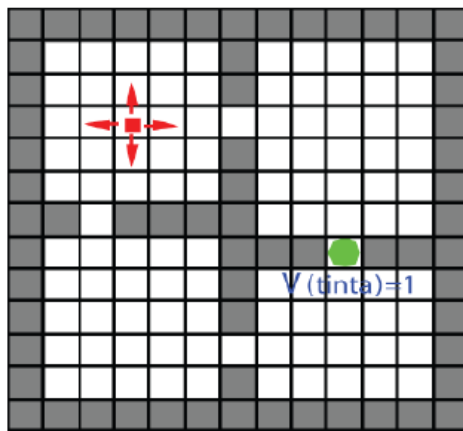
- Învățare supervizată
- Învățare ne-supervizată
- **Învățare cu întărire**
- Teoria învățării

Învățare cu întărire

- Scop
 - Învățarea, de-a lungul unei perioade, a unui mod de acțiune (comportament) care să maximizeze recompensele (câștigurile) pe termen lung
- Tip de probleme
 - Ex. Dresarea unui câine (good and bad dog)
- Caracteristic
 - Interacțiunea cu mediul (acțiuni → recompense)
 - Secvență de decizii
- Învățare supervizată
 - Decizie → consecință (cancer malign sau benign)

Învățare cu întărire – definiere

- Exemplu: plecând din căsuța roșie să se găsească un drum până la căsuța verde



- Agentul învață prin interacțiunea cu mediul și prin observarea rezultatelor obținute din aceste interacțiuni
 - Este vorba de "cauză și efect" -- modul în care oamenii își formează cunoașterea asupra mediului pe parcursul vieții
 - Acțiunile pot afecta și recompensele ulterioare, nu numai pe cele imediate (efect întârziat)

Învățare cu întărire – definire

Învățarea unui anumit comportament în vederea realizării unei sarcini → execuția unei acțiuni → primește un feedback (cât de bine a acționat pentru îndeplinirea sarcinii) → execuția unei noi acțiuni

Învățare cu întărire

- Se primește o recompensă (întărire pozitivă) – dacă sarcina a fost bine îndeplinită
- Se primește o pedeapsă (întărire negativă) – dacă sarcina nu a fost bine îndeplinită

Definire

- Se dau
 - Stări ale mediului
 - Acțiuni posibile de executat
 - Semnale de întărire (scalare) – recompense sau pedepse
- Se determină
 - O succesiune de acțiuni care să maximizeze măsura de întărire (recompensa)

Alte denumiri

- Reinforcement learning
- Învățare împrăștiată

Învățare cu întărire – definire

- Învățare supervizată
 - Învățarea pe baza unor exemple oferite de un expert extern care deține o bază importantă de cunoștințe
- Învățare cu întărire
 - Învățarea pe baza interacțiunii cu mediul

Învățare cu întărire – exemple

□ Robotică

- Controlul membrelor
- Controlul posturii
- Preluarea mingii în fotbalul cu roboții

□ Cercetări operaționale

- Stabilirea prețurilor
- Rutare
- Planificarea task-urilor

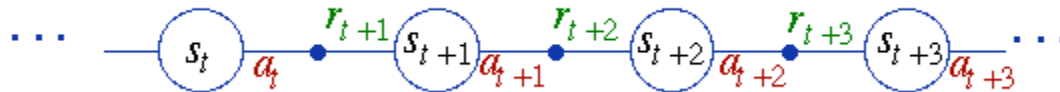
Învățare cu întărire – proces

Procesul

- Agentul observă o stare de intrare
- Agentul alege o acțiune pe baza unei funcții de decizie (o **strategie**)
- Agentul execută acțiunea aleasă
- Agentul primește o recompensă/pedeapsă numerică de la mediu
- Agentul reține recompensa/pedeapsa primită

Învățare cu întărire – proces

- Mediul este modelat ca un proces de decizie de tip Markov
 - S – mulțimea stărilor posibile
 - $A(s)$ – acțiuni posibile în starea s
 - $P(s, s', a)$ – probabilitatea de a trece din starea s în starea s' prin acțiunea a
 - $R(s, s', a)$ – recompensa așteptată în urma trecerii din starea s în starea s' prin acțiunea a
 - γ – rata de discount pentru recompensele întârziate



- Obiectivul
 - Găsirea unei politici $\pi : s \in S \rightarrow a \in A(s)$ care maximizează
 - valoarea (recompensa viitoare așteptată) a unei stări
 - $V^\pi(s) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \mid s_t = s, \pi\}$
 - calitatea fiecărei perechi stare-acțiune
 - $Q^\pi(s, a) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \mid s_t = s, a_t = a, \pi\}$

Învățare cu întărire – evaluare

Măsuri de performanță

- Recompensa acumulată pe parcursul învățării
- Numărul de pași necesari învățării

Învățare cu întărire – algoritmi

- Q-learning
 - Calitatea unei combinații stare-acțiune
- SARSA (*State-Action-Reward-State-Action*)

Învățare automată

□ Instrumente

- Weka <https://www.cs.waikato.ac.nz/ml/weka/>
- Scikit-Learn <http://scikit-learn.org/stable/>
- Pattern <https://www.clips.uantwerpen.be/pattern>
- Rapid Miner <https://rapidminer.com/>
- Orange <https://orange.biolab.si/>