Calcul neuronal si evolutiv.

Lab 1: Probleme de clasificare
Platforma Weka pentru explorarea datelor
Date de test pentru metode de învățare automată (Machine Learning Repository)
Familiarizare cu Scilab și R

---

1. Probleme de clasificare

Problemele de clasificare necesită gruparea unor obiecte descrise prin caracteristicile lor (vectori de trăsături sau de atribute) în clase predefinite. Clasificatorii se construiesc pornind de la exemple de clasificare corectă printr-un proces de învățare (numită învățare supervizată spre deosebire de cazul învățării nesupervizate unde clasele nu sunt predefinite).

Problema clasificării intervine în numeroase domenii şi numeroase probleme practice pot fi formulate ca probleme de clasificare. Exemple de probleme de clasificare:

- *Recunoașterea caracterelor*:
  - Datele de intrare conțin caracteristici ale caracterelor (obținute în urma preprocesării reprezentării grafice a caracterului)
  - Clasele corespund caracterelor care urmează a fi identificate (litere, cifre, alte simboluri)

- *Identificarea mesajelor de tip spam*:
  - Datele de intrare sunt informații obținute prin preprocesarea mesajelor (informațiile obținute prin preprocesare se extrag atât din antetul mesajului cât și din conținutul acestuia și se referă de regulă la frecvența de apariție a unor cuvinte sau construcții sintactice stocate anterior într-o colecție de elemente considerate ca fiind potențial suspecte)
  - Clasele sunt reprezentate de cele două categorii: mesaje legitime, respectiv mesaje frauduloase (de tip spam)

- *Diagnoza medicală*:
  - Datele de intrare sunt simptome, rezultate ale investigațiilor, caracteristici ale pacientilor etc.
  - Clasele sunt corelate cu diagnosticul (în cazul binar poate fi vorba de absența sau prezența unei boli)

2. Tehnici de clasificare

Tehnicile de clasificare au ca scop principal extragerea unor modele de clasificare (clasificatoare) pornind de la un set de date care ilustrează legătura dintre obiectele de clasificat și clasele corespunzătoare. Modelele de clasificare pot fi binare (în cazul clasificării în două clase) sau multiple (în cazul clasificării în mai multe clase).

In funcție de modul in care se extrage modelul de clasificare şi de structura finală există mai multe categorii de clasificatoare:

- *Arbori de decizie.* Nodurile interne ale arborelui conțin condiții referitoare la valorile atributelor, iar nodurile frunză conțin etichete ale claselor; clasificarea unui obiect presupune parcurgerea unei căi în arbore pornind de la rădăcină până la un nod frunză – nodul frunză la care se ajunge indică clasa.
- *Reguli de decizie.* Sunt construcții de tip IF-THEN care conțin în antecedent expresii logice implicând atributele iar in partea de concluzie eticheta unei clase; clasificarea presupune parcurgerea setului de reguli și identificarea regulii care se potrivește cu datele.
- *Clasificatoare bazate pe principiul celui mai apropiat vecin (nearest neighbor).* Clasificatorul constă dintr-un set de exemple de clasificare; clasificarea constă în determinarea pentru un un obiect de clasificat a celor mai apropiate exemple din set și selecția clasei majoritare.
- *Rețele Bayesiene.* Sunt modele probabiliste descrise prin grafuri care surprind relațiile dintre clase și atribute; nodurile din graf sunt asociate fie atributelor (interpretate ca variabile observabile) fie unor variabile latente, iar arcele sunt associate corelațiilor dintre entități. Clasificarea se bazează pe estimarea probabilității fiecărei clase, condiționată de datele de intrare.
- *Rețele neuronale.* Sunt structuri de tip graf care modelează dependența neliniară dintre atributele de intrare și clase. Antrenarea presupune determinarea paramentrilor modelului neliniar pornind de la exemple de antrenare, iar clasificarea se bazează pe determinarea semnalului de ieșire și stabilirea clasei pe baza acestuia.
- *Clasificatoare bazate pe vectori support (Support Vector Machines).* Sunt modele de clasificare bazate pe identificarea unor hiperplane care asigură separarea claselor. Antrenarea presupune estimarea parametrilor hiperplanelor (și eventual a unor funcții nucleu utilizate în transformarea problemelor neliniar separabile în problem linear separabile) iar clasificarea constă în calculul valorii asociate funcției (funcțiilor) separatoare.

2. Etapele proiectării unui clasificator

a) *Pregătirea datelor.* Această etapă presupune pre-procesarea datelor (de exemplu eliminarea erori, tratarea valorilor absente, normalizarea etc.) și distribuirea acestora în trei seturi principale:

   - *Date de antrenare*: date utilizate în procesul de antrenare pentru determinarea parametrilor clasificatorului (în cazul rețelelor neuronale se determina ponderile conexiunilor dintre neuroni)
   - *Date de validare*: date utilizate pentru a analiza comportamentul clasificatorului pe parcursul algoritmului de învățare; performanța obținută pe setul de validare în timpul procesului de învățare este utilizată pentru a decide dacă învățarea trebuie continuată sau nu.
   - *Date de testare*: sunt utilizate pentru a analiza performanțele unui clasificator antrenat.

b) *Antrenarea clasificatorului.* Presupune extragerea modelului de clasificare din date. Modul de antrenare depinde de tipul de clasificator. In cazul arborilor de decizie, prin antrenare se identifică ordinea în care se selectează atributele datelor pentru a fi asignate nodurilor și a realiza ramificarea. În cazul rețelelor neuronale antrenarea permite determinarea parametrilor asociați conexiunilor dintre neuroni (ponderile conexiunilor).

c) *Evaluarea clasificatorului.* Pentru evaluarea calității antrenării se poate folosi tehnica *validării încrucișate* (cross-validation) care constă în divizarea setului inițial de date într-un număr de S subseturi și repetarea de S ori a procesului de antrenare de fiecare dată utilizând alt subset de antrenare. La fiecare repetare antrenarea se bazează pe S-1 subseturi iar validarea se face utilizând subsetul care nu a fost utilizat la antrenare.

Calitatea unui clasificator din perspectiva identificării corecte a unei clase se măsoară folosind informațiile din *matricea de confuzie* care conține:

- Numărul de date clasificate corect ca aparținând clasei de interes: True positive cases (TP)
- Numărul de date clasificate corect ca neaparținând clasei de interes: True negative cases (TN)
- Numărul de date clasificate incorect ca aparținând clasei de interes: False positive cases (FP)
- Numarul de date clasificate incorect ca neaparținând clasei de interes: False negative cases (FN)

In cazul clasificării în două clase P(ozitiv) și N(egativ) matricea de confuzie are structura:

	Clasa prezisă: P	Clasa prezisă: N
Clasa reală: P	TP	FN
Clasa reală: N	FP	TN

Pe baza acestor valori se pot calcula o serie de alte măsuri: acuratețe, rata de eroare, senzitivitate (capacitatea clasificatorului de a identifica toate cazurile pozitive), specificitate (capacitatea clasificatorului de a nu identifica ca pozitive cazuri care sunt in realitate negative), precizie și regăsire:

$accuracy = (TP+TN)/(TP+FP+TN+FN)$ $errorRate=(FP+FN)\ )/(TP+FP+TN+FN)$

$sensitivity = TP\ /\ (TP + FN)$ $specificity = TN/(TN+FP)$

$precision = TP/(TP+FP)$ $recall = TP/(TP+FN)=sensitivity$

$F=2*precision*recall/(precision+recall)$

Specificitatea și senzitivitatea sunt folosite de regulă în analiza datelor biomedicale, iar precizia și gradul de regăsire sunt utilizate în analiza textelor (text mining).

O altă modalitate de a evalua calitatea unui clasificator binar este reprezentată de curba ROC (Receiver Operator Characteristic) care conține puncte având coordonatele *(1-specificity, sensitivity)* și care corespund la valori diferite ale pragului de decizie. Pragul de decizie este folosit pentru a decide pe baza valorii produse de neuronul aferent clasei de interes dacă data de intrare aparține sau nu clasei (dacă valoarea de ieșire a neuronului este mai mare decât pragul atunci data aparține clasei, altfel nu aparține). Ideal este ca aria suprafeței aflată sub curba ROC sa fie cât mai apropiată de 1.

3. Utilizarea rețelelor neuronale în rezolvarea problemelor de clasificare

Pentru rezolvarea problemelor de clasificare, arhitectura cea mai frecvent folosită este cea feedforward caracterizată prin:

- Un *nivel de intrare* având atâtea unități câte componente (atribute) au datele de intrare
- Unul sau mai multe *nivele ascunse* (cu cât numarul de unități ascunse este mai mare cu atât modelul extras de rețea este mai complex; acest lucru poate fi un dezavantaj conducând la diminuarea capacității de generalizare a rețelei).
- Un *nivel de ieșire* având atâtea unități cât este numărul de clase

Semnalul de ieșire corespunzător unui semnal de intrare (vector de caracteristici asociate obiectului de clasificat) indică clasa căreia îi aparține obiectul de clasificat. Variante de interpretare:

- In cazul general, indicele unității pentru care semnalul de ieșire este maxim corespunde clasei
- In cazul în care valorile produse de către unitățile de ieșire sunt cuprinse în intervalul (0,1) și suma lor este egală cu 1 atunci pot fi interpretate ca probabilități de apartenență la fiecare clasă.

4. Platforma Weka pentru explorarea datelor și învățare automată

(http://www.cs.waikato.ac.nz/ml/weka/)

- Weka este o colecție de algoritmi pentru analiza datelor și învățare automată (incluzând câteva tipuri de rețele neuronale: perceptron multinivel, rețea cu functii radiale, clasificatori cu vectori suport) proiectate să rezolve probleme de analiză a datelor: vizualizare, selecție atribute, clasificare, grupare, extragere reguli de asociere.

- Este o colecție open-source și metodele pot fi apelate atât din interfața grafică cât și din cod Java. Există trei variante principale de interfață: Explorer (pentru efectuarea de prelucrări individuale asupra unui set de date), Experimenter (pentru compararea performanțelor mai multor metode pe același set de date) și Knowledge Flow (pentru definirea unui flux de prelucrari).

Exercitiul 1. Deschideți fișierul "breast_cancer_nominal.csv" în Weka și comparați acuratețea următorilor clasificatori:

- Reguli de clasificare: Rules-> OneR, Rules-> ConjunctiveRules, Rules->NNge
- Arbori de decizie: Trees->J48, Trees->RandomForest
- Cel mai apropiat vecin: Lazy->IBk
- Retele bayesiene: bayes->NaiveBayes
- Retele neuronale: functions->MultilayerPerceptron, functions->RBFnetwork,
- Clasificatori bazați pe vectori suport: functions->SMO

Indicație: construiți un fișier Excel care conține pe o coloana tipul clasificatorului iar pe cealaltă coloana valoarea acurateței ("correctly classified instances").

5. Machine Learning Repository (http://archive.ics.uci.edu/ml/)

Arhiva conține numeroase seturi de date din diferite domenii (medicină, fizică, informatică si inginerie) care pot fi utilizate pentru antrenarea, validarea și compararea clasificatorilor și a altor metode de învățare automată.

Exercitiul 2. Vizitați colecția, descărcați unul dintre seturile de date pentru clasificare (la alegere) și utilizați-l pentru analiza metodelor de clasificare implementate în Weka.

Anexa 1: Scilab - Open source software for numerical computation
(http://www.scilab.org/)

Scilab este un limbaj de programare interpretat care oferă suport pentru prelucrări specifice din algebra liniară, pentru prelucrări asupra funcțiilor polinomiale și a celor raționale, interpolare și aproximare, optimizare liniară, optimizare pătratică, rezolvarea ecuațiilor diferențiale, prelucrarea semnalelor, statistică și prelucrări grafice. Ca principiu de lucru dar și ca interfață (în ultimele versiuni) este similar pachetului Matlab.

In Scilab, la fel ca în Matlab, obiectul principal este *matricea*, atât vectorii cât și scalarii fiind considerați cazuri particulare de matrici.

Aspecte generale:

- Scilab este case-sensitive
- Fiind interpretor, variabilele nu trebuie declarate însă trebuie să aibă asignată o valoare; operatorul de asignare se sepcifică prin =
- Constantele predefinite au numele prefixat de % (de exemplu: %pi, %i, %e, %t (true), %f (false))
- Operatorii relaționali sunt: == (egal), ~= sau <> (diferit), <=, >=
- Operatorii logici sunt: ~ (not), & (and), | (or)
- Rezultatul unei evaluări este asignat implicit variabilei ans care poate fi utilizată în comanda imediat următoare
- Comenzile plasate pe aceeași linie trebuie separate prin ; (separatorul ; are și efectul inhibării vizualizării rezultatului ultimei evaluări)
- Șirurile de caractere se încadrează între ghilimele (") iar operatorul de concatenare este +
- Comentariile de tip linie se specifică prin //

Specificarea/definirea matricilor.

- Explicit prin enumerarea elementelor (elementele de pe aceeași linie se separă prin virgulă (sau spațiu) iar liniile se separă prin punct-virgula (sau enter):
  A=[a11,a12,…a1n; a21,a22,…,a2n; …;am1, am2,…,amn]

- Implicit prin utilizarea unor funcții care generează matrici:
  - zeros(m,n) : matrice cu m linii și n coloane și elemente egale cu 0
  - ones(m,n): matrice cu m linii și n coloane și elemente egale cu 1
  - rand(m,n): matrice cu m linii și n coloane și elemente generate uniform aleator în intervalul (0,1)

Operații asupra matricilor.

- *Determinarea dimensiunii:* size(matrice) returnează vectorul [nr linii, nr coloane]
- *Reorganizarea unei matrici:* matrix(matrice,nrlinii,nrcoloane) returnează o matrice cu numărul de linii și de coloane specificat și elementele din matricea inițială
- *Specificarea elementelor:* matrice(indice linie,indice coloana). Obs: indicii pot atât valori individuale cât și domenii de valori specificate prin inf:sup . Pentru a specifica ultimul indice de linie/coloană se poate utiliza $. De exemplu matrice($,$-1) indică elementul de pe ultima linie, penultima coloană.
  Obs: indicii elementelor încep cu 1
- *Modificarea unei matrici:*
  - Modificare element: matrice(i,j)=valoare

- Adăugare linie: matrice=[matrice; el1, el2,…,eln]
- Adăugare coloana: matrice=[matrice'; el1, el2,…,elm]' (operatorul ' corespunde calculului transpusei matricii)
- Ștergere linie: matrice(i,:)=[]
- Ștergere coloană: matrice(:,j)=[]

- *Operații aritmetice:* toate operațiile aritmetice sunt vectorizate; pentru a efectua operații la nivel de element trebuie specificat . (punct) înainte de operator. De exemplu A*B returnează produsul matricilor A și B iar A.*B returnează matricea în care pe linia i coloana j se află produsul elementelor aflate pe aceeași poziție în cele două matrici. Este indicat să fie folosite operații vectorizate în locul iterării explicite a unor operații scalare întrucât sunt mai eficiente (de la 10 până la 100 ori mai rapide).

Alte tipuri de obiecte în Scilab:

- *Structuri:*
  - struct(numeCâmp1,valoare1,numeCâmp2,valoare2,….,numeCâmpn, valoaren)
  - Exemplu: data=struct('zi',30,'luna','septembrie','an',2014)
  - Specificare elemente: numeStructura.numeCâmp (de exemplu: data.zi este 30)

- *Liste heterogene:*
  - *Listă simplă*: list(Element1,Element2,…,Elementn)
    Obs: elementele se specifică utilizând indici
  - *Listă cu tip*: tlist(listaNumeElemente,Element1,Element2,…,Elementn) ;
    Exemplu: d=tlist(['data','zi','luna','an'],20,'sept',2014)
    Obs: elementele pot fi specificate atât prin indiciere cât și prin calificare: d(2) este identic cu d.zi

Instrucțiuni de control (ramificare și ciclare)

- **Instrucțiunea if:**

```
if (conditie) then
        <Prelucrare 1>
else
        <Prelucrare 2>
end
```

Varianta imbricată:

```
if (conditie1) then
        <Prelucrare 1>
elseif (conditie 2)
        <Prelucrare 2>
else
        <Prelucrare 3>
end
```

- **Instrucțiunea select:**

```
select <variabila selector>
case <valoare 1>
```

```
        <prelucrare 1>
case <valoare 2>
        <prelucrare 2>
…
case <valoare n>
        <prelucrare n>
else
        <alta prelucrare>
end
```

- **Instrucțiunea for**

```
for contor=inf:pas:sup
        <Prelucrare>
end
```

Obs: Dacă pasul este 1 atunci poate fi omis din specificarea domeniului. Valoarea pasului poate fi și număr negativ. Iterarea se poate face și pe elementele unui vector (matrice linie):

```
for contor=vector
        <Prelucrare>
end
```

- **Instrucțiunea while**

```
while(conditie)
        <Prelucrare>
end
```

Definirea și apelul funcțiilor

Funcțiile pot fi utilizate pentru a calcula mai multe rezultate (specificate prin variabilele de ieșire indicate la definirea funcției):

```
function [output1, …,outputm]=numeFunctie(input1,…,inputn)
        <corp functie>
endfunction
```

Prelucrări grafice

Principalele funcții grafice din Scilab sunt:

- plot - pentru reprezentarea grafică a funcțiilor uni-dimensionale
- fplot3d și contour - pentru reprezentarea grafică a suprafețelor
- paramfplot2d – reprezentare curbe descries prin ecuații parametrice
- polarplot – reprezentare în coordonate polare

Exemple:

```
// graficul unei functii
function y=f(x)
  y=x*x/10+sin(x)
endfunction
```

```
x=-2*%pi:0.1:2*%pi
clf
plot(x,f)

// graficul unei suprafete
function y=f2arg(x1, x2) // function with 2 arguments
   y = x1 **2 + x2 **2;
endfunction
x1data = linspace ( -1 , 1 , 100 ); // this is equivalent with x1data = -1:0.1:1;
x2data = linspace ( -1 , 1 , 100 );
contour ( x1data , x2data , f2arg , 10)  // contour plot
pause
clf                              // clean the screen
fplot3d( x1data , x2data , f2arg)       // surface plot
```

ANN Toolbox - Pachet de rețele neuronale în Scilab
(https://atoms.scilab.org/toolboxes/ANN_Toolbox)

Informații generale:

- creat de Ryurick M. Hristev și Allan Cornet
- implementează rețele feedforward multinivel și mai multe variante ale algoritmului BackPropagation:
  - standard
  - varianta cu moment
  - varianta bazată pe metoda gradientului conjugat
  - varianta SuperSAB (self-adaptive Backpropagation); se bazează pe adaptarea ratei de învățare separate pentru fiecare pondere – permite accelerarea antrenării

Utilizare:

- se dezarhivează ANN_Toolbox
- se execută builder.exe și loader.exe (prin Open în SciNotes și Execute)

Exemplu: implementarea unei rețele pentru recunoaștere de semne grafice (adaptat după exemplul de la http://burubaxair.wordpress.com/tag/neural-network/)

```
// specificarea semnelor grafice din setul de antrenare
T = [...
1 1 1 1 1 ...
0 0 1 0 0 ...
0 0 1 0 0 ...
0 0 1 0 0 ...
0 0 1 0 0 ...
0 0 1 0 0 ...
0 0 1 0 0 ...
]';
U = [...
1 0 0 0 1 ...
1 0 0 0 1 ...
1 0 0 0 1 ...
1 0 0 0 1 ...
```

```
1 0 0 0 1 ...
1 0 0 0 1 ...
0 1 1 1 0 ...
]';
// arhitectura retelei: 35 unitati de intrare, 10 unitati ascunse, 2 unitati de iesire
N = [35 10 2];
// initializarea ponderilor retelei
W = ann_FF_init(N);
// construirea setului cu date de intrare
x = [T, U];
// construirea setului cu raspunsuri corecte
t_t = [1 0]';
t_u = [0 1]';
t = [t_t, t_u];
// specificarea ratei de invatare si a tolerantei la eroare
lp = [0.01, 1e-4];
// specificarea numarului maxim de epoci de antrenare
epochs = 3000;
// antrenarea retelei
W = ann_FF_Std_batch(x,t,N,W,lp,epochs);
// verificarea comportarii pe setul de antrenare = simularea retelei
y = ann_FF_run(x,N,W)
// afisarea rezultatului simularii
disp(y)
```

Exercitiu 3. Modificați secvența Scilab anterioară pentru:

a) testarea funcționării pentru variante alterate de zgomot ale simbolurilor U și T
b) extinderea clasificatorului pentru mai multe simboluri (clase), de exemplu A și L
c) utilizarea altor variante ale algoritmului de antrenare:

- varianta cu moment.

  Indicație: `lp = [0.01, 1e-4,0.9,0.1]; W=ann_FF_Mom_batch(x,t,N,W,lp,epochs)`
- variante online (`ann_FF_Std_online , ann_FF_Mom_online`)

Tema: proiectați o rețea neuronală, folosind Scilab-ANN, pentru clasificarea datelor din fișierul `breast_cancer_nominal.csv`

Anexa 2: Pachetul R (R free software environment for statistical computing and graphics - http://www.r-project.org/)

R este o suită integrată de instrumente software destinate prelucrării datelor, efectuării unor calcule și vizualizării grafice. R conține o colecție mare de pachete destinate diferitelor tipuri de prelucrări (în principal de natură statistică) și un limbaj care permite extinderea facilităților prin definirea de noi funcții și pachete.

Caracteristici ale limbajului R.

Aspecte generale:

- Este case-sensitive
- Cea mai simplă și frecvent utilizată comandă (de atribuire) se specifică prin
  nume_ variabila <- valoare sau
  nume_ variabila = valoare sau
  assign("nume_variabila",valoare)
- Comenzile (instrucțiunile) se separă prin ; sau se plasează pe linii separate
- Comenzile elementare pot fi grupate prin specificare între { și }
- Comentariile se specifica prin #
- Execuția comenzilor stocate într-un fisier se face prin
  source("fisier_comenzi.R")

Obiecte în R:

- *Scalari* (valori individuale care pot fi interpretate ca vectori cu un singur element):
  - Valori numerice (întregi, reale, complexe)
    - Conversii între tipuri: var2=as.tip_nou(var1) (ex: as.character(2) returneaza "2")
  - Logice (constante predefinite: TRUE sau T, FALSE sau F)
    - Operatori logici: ! (negatie), | (disjunctie), & (conjunctie)
    - Operatori relaționali: <,>,<=,>=,== (egalitate),!= (diferit)
  - Simboluri
  - Constante uzuale:
    - NA (valoare nedefinita)
    - pi
- *Vectori* (colecții omogene de valori accesibile prin indice de poziție).
  - Construire vector: nume_vector<-c(el1,el2,...,eln)
  - Accesare element: nume_vector[indice] (indicii sunt numerotați începând de la 1)
  - Operații:
    - operațiile aritmetice (+,-,*,/) și funcțiile matematice unare (exp, log, sqrt,sin,cos etc.) aplicate asupra vectorilor acționează la nivelul fiecărui element (cei doi vectori trebuie să aibă același număr de elemente)
    - determinarea numărului de elemente din vector: length(vector)
    - determinare minim respectiv maxim: min(vector) respectiv max(nume_vector)
    - calcul suma respectiv produs elemente: sum(vector) respectiv prod(vector)

        - sortare crescătoare: sort(vector)
        - selecție elemente: nume_vector[secventa_indici] (ex: vector[c(1,3,5)] returnează elementele de pe pozițiile 1, 3 si 5); secvența de indici poate fi creata prin orice funcție (ex: vector[vector>0] returneaza vectorul continand doar elementele pozitive din vectorul initial.

    - Generarea unor secvențe de valori:
        - valoare_initiala:valoare_finala (ex: 1:5 genereaza vectorul corespunzator lui c(1,2,3,4,5))
        - folosind functia seq(valoare_initiala, valoare_finala, by pas) (ex: seq(1,5,by 1))

- *Liste* (colecții nu neaparat omogene de obiecte accesibile prin indice sau nume)
    - Construire lista: lista=list(element1, element2,...); fiecare element al listei poate fi un obiect anonim sau cu nume
      (ex: lista1=list("zi"=30,"luna"="septembrie","an"=2014))
    - Accesarea valorilor elementelor:
        - Prin indici: lista[[indice]] (ex: lista1[[1]] va returna 30) (Obs: dacă se folosește lista[indice] se returnează atât valoarea cât și numele elementului.
        - Prin nume: lista$nume_element (ex: lista1$an va returna 2014)
    - Concatenarea listelor: c(lista1,lista2,...)
    - Modificarea unei liste: lista[[indice]]=element_nou
- *Tablouri (uni și multi-dimensionale)*
    - Construire: tablou=array(vector_valori,dim=c(dim1,dim2, ...)
      ex: matrice=array(1:6,c(2,3))
    - Specificare elemente: tablou[indice1,indice2,...] (Obs: există multe variante de a accesa subseturi de elemente aflate pe diferite poziții în matrice)
    - Operațiile aritmetice clasice (+,*,-,/) se pot aplica asupra unor tablouri fiind mapate la nivel de element (ca în cazul vectorilor). Pentru a calcula produsul a două matrici (calculând produsul scalar al liniilor si coloanelor corespunzatoare) se poate folosi operatorul %*%
- *Tabele de frecvențe*
    - Construire: tabel_frecv=table(vector) (ex: table(c(2,1,3,1,2)))

- *Tabele de date* (Data frame): structuri bidimensionale ce conțin mai multe înregistrări specificate ca linii numerotate, fiecare coloană corespunzând unui câmp (atribut) din înregistrare și având asociat un nume. Tabelele de date se pot construi explicit folosind funcția data.frame sau pot fi completate cu date din fișiere folosind funcția read.table
  Exemplu: apelul data.frame(x=seq(1,20,2),y=1:10) generează:

	x	y
1	1	1
2	3	2
3	5	3
4	7	4
5	9	5
6	11	6
7	13	7
8	15	8
9	17	9
10	19	10

Structuri de control

- Prelucrări decizionale: if (conditie) instructiune1 else instructiune 2
- Prelucrări repetitive:
  - For: for (contor in domeniu) instructiune
    (obs: în varianta cea mai simplă domeniul se specifică prin valoarea_initiala:valoare_finala)
  - While: while(conditie) instructiune
  - Repeat: repeat instructiune (întreruperea ciclului repeat se poate face doar prin break).

Definirea funcțiilor

- Definirea unei funcții: nume_functie = function(arg1, arg2, …){instructiuni; return(rez)}
- Apelul unei funcții: nume_functie(val1,val2,….) (Obs: la apel în loc de a specifica doar valoarea unui argument (val1) se poate specifica o pereche de forma nume_argument=valoare (ex: arg1=val1). In acest caz ordinea parametrilor de apel nu trebuie să coincidă neapărat cu ordinea specificată la definirea funcției.
- Particularități:
  - Variabilele utilizate în cadrul unei funcții sunt implicit considerate variabile locale astfel că efectele operațiilor de asignare specificate cu <- sau = au doar efect local
  - Pentru a efectua o asignare cu caracter global (al cărui efect să rămână după revenirea din apel) se poate folosi operatorul de "superasignare": <<-

Prelucrări grafice simple

- Graficul unei funcții: plot(nume_functie, limita_inferioara, limita_superioara) (ex: plot(sin,0,pi))
- Grafic liste puncte: plot(vector_abscise,vector_ordonate) (ex: plot(a,a^2) cu a un vector cu valori numerice)

Citirea datelor din fișiere

- Citire sub forma de obiecte de tip "data frame" (tabele de valori în care prima linie conține denumiri ale câmpurilor iar prima coloană conține numărul de ordine al înregistrării). Este adecvată pentru citirea din fișierele ce conțin înregistrări specificate pe câte o linie.
  - Apel: read.table("nume_fisier")

- Citire sub forma de vector cu elemente de tip vector (în cazul în care fișierul conține linii cu același număr de valori separate prin spații dar nu conține o linie cu antet). Un element al vectorului returnat corespunde unei coloane din fișier (rezultatul nu e o structură de tip matrice astfel că pentru a accesa a treia valoare din linia 2 trebuie specificat rez[[3]][[2]].
  - Apel: scan("nume_fisier",lista_tipuri) (ex: scan("fisier.dat",list("",0)) – citeste un fișier cu două coloane în care pe prima coloană se află caractere iar pe a doua valori numerice

Pachete R pentru învățare automata: http://cran.r-project.org/web/views/MachineLearning.html

Pachete R pentru implementarea rețelelor neuronale feedforward:

- nnet = software for feed-forward neural networks with a single hidden layer, and for multinomial log-linear models.
- neuralnet = training of neural networks using backpropagation, resilient backpropagation with or without weight backtracking

Detalii utilizare nnet

Apel:

nnet(x, y, weights, size, Wts, mask, linout=FALSE, entropy=FALSE, softmax=FALSE, censored=FALSE, skip=FALSE, rang=0.7, decay=0, maxit=100, Hess=FALSE, trace=TRUE, MaxNWts=1000, abstol=1.0e-4, reltol=1.0e-8)

Semnificația parametrilor:

x: matrice sau "data frame" conținând datele de intrare din setul de antrenare
y: matrice sau "data frame" conținând datele de ieșire (răspunsurile corecte) din setul de antrenare
weights: ponderi associate exemplelor din setul de antrenare (dacă nu se specifică se consider 1)
size: numărul unităților de pe nivelul ascuns
wts: valori inițiale ale ponderilor (dacă nu se specifică se aleg aleator)
mask: vector binary prin care se specific care dintre parametric vor fi optimizați (implicit se optimizează toți parametrii)
linout: unitățile de ieșire au funcții liniare de activare (implicit toate funcțiile de activare sunt logistice)
entropy: se folosește entropia ca și criteriu de optimizare (implicit se folosește suma pătratelor erorilor)
softmax și censoring: se folosesc pt modele log-liniare
skip: se include și conexiuni directe între nivelul de intrare și cel de ieșire
rang: [-rang,rang] indică domeniul de inițializare al ponderilor; valoarea indicată este în jur de 0.5 (sau astfel încât rang*max(|x|) este cca 1)
decay: parametru de penalizare a ponderilor mari (vezi varianta cu termen de regularizare din Curs 2)
maxit: numărul maxim de epoci de antrenare (implicit: 100)
hess: se returnează matricea hessiana corespunzătoare celui mai bun set de valori ale ponderilor
trace: se afișează rezultate intermediare pe parcursul procesului de antrenare
maxNWts: numărul maxim de ponderi
abstol: toleranța la antrenare (procesul de antrenare este oprit dacă valoarea erorii devine mai mica decât abstol)
reltol: procesul de antrenare se oprește dacă eroarea nu descrește cu un factor de cel puțin 1-reltol de la o epocă la alta

Funcția nnet returnează un obiect de tip nnet din care în mod usual sunt consultate următoarele componente:

wts: valorile ponderilor după antrenare
value: valoarea criteriului de optimizat după antrenare

fitted.values: rezultatele corespunzătoare exemplelor din setul de antrenare
residuals: eroarea după antrenare
convergence: 1 (dacă s-a atins numărul maxim de epoci), 0 (altfel)

Exemplu de utilizare nnet:

```
# extragere date pentru setul de antrenare din setul implicit Iris
ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3])  #date de intrare
targets <- class.ind( c(rep("s", 50), rep("c", 50), rep("v", 50)) )              # raspunsuri corecte
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))    # selectie set antrenare
irisNnet <- nnet(ir[samp,], targets[samp,], size = 2, rang = 0.1,decay = 5e-4, maxit = 200)   #antrenare retea

# functie pentru construirea matricii de confuzie
test.cl <- function(true, pred) {
        true <- max.col(true)
        cres <- max.col(pred)
        table(true, cres)
}

# vizualizare matrice de confuzie calculate folosind datele care nu au fost incluse in setul de antrenare
test.cl(targets[-samp,], predict(irisNnet, ir[-samp,]))
```

Observație: datele de intrare și cele de ieșire pot fi specificate utilizând o "formula" în care se face referire la numele coloanelor dintr-un tabel de date sub forma:

variabile ieșire ~ variabila intrare 1 + variabila intrare 2 ...

Exemplu:

```
ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),species = factor(c(rep("s",50), rep("c", 50), rep("v", 50))))

# formula species ~ . indică faptul ca rețeaua are ca scop descoperirea asocierii dintre specie si toate
# atributele datelor
irisNnet2 <- nnet(species ~ ., data = ird, subset = samp, size = 2, rang = 0.1,decay = 5e-4, maxit = 200)

table(ird$species[-samp], predict(irisNnet2, ird[-samp,], type = "class"))
```