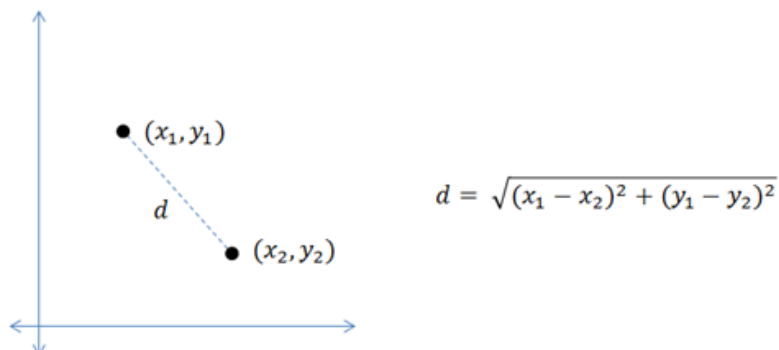


## Gruparea datelor folosind K-means

Gruparea (engl. *clustering*) este una dintre cele mai frecvent utilizate tehnici de analiză exploratorie a datelor, ideea fiind de a deduce regulile și modalitatea de structurare a acestora. Metodele de clusterizare presupun identificarea unor configurații de grupare a datelor astfel încât grupurile formate să fie constituite din elemente cu un anumit grad de similaritate. Altfel spus, se dorește gruparea datelor în așa fel încât în cadrul aceluiași grup datele să fie similare într-o mare măsură, în timp ce datele din grupuri diferite să fie, la rândul lor, diferite. Gruparea presupune stabilirea unei modalități de evaluare a similarității a două elemente din setul de date ce se grupează. Măsurarea similarității se realizează prin stabilirea unei metrici corespunzătoare (o metodă de cuantificare a similarității a două elemente). Metrica se decide funcție de specificul datelor și de scopul grupării, de exemplu poate fi distanța euclidiană, o distanță bazată pe corelație, sau o formulă mai complexă care ia în calcul o multitudine de trăsături ale datelor.

Exemple de metrici de similaritate:

- distanța euclidiană dintre două puncte:



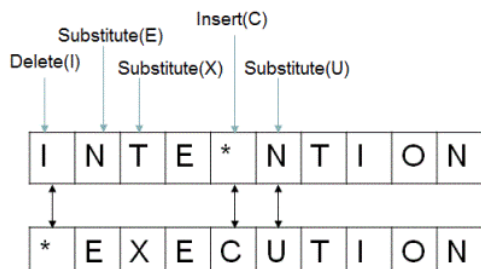
- similaritatea cosinus dintre doi vectori:

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

$$\|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + a_3^2 + \dots + a_n^2}$$

$$\|\vec{b}\| = \sqrt{b_1^2 + b_2^2 + b_3^2 + \dots + b_n^2}$$

- distanța Levenshtein dintre două șiruri de caractere (numărul de operații de inserare/ștergere/substituție necesare pentru a obține unul dintre șiruri pornind de la celălalt)



K-means este unul dintre cei mai simpli algoritmi de grupare. În cadrul grupării K-means se încearcă partiționarea setului de date în K grupuri (clustere) distincte care nu se suprapun, unde fiecare element al setului de date aparține unui singur grup (există alte metode de grupare unde un element poate aparține mai multor grupuri cu anumite probabilități). Numărul de grupuri K este prestabilit. Scopul este de a crea grupuri în cadrul cărora elementele să fie cât mai similare posibil, în timp ce elementele din grupuri diferite să fie cât mai puțin similare posibil. Cu cât există mai puține variații în cadrul grupurilor, cu atât punctele de date sunt mai omogene (similare) în cadrul aceluiași grup. Cu cât variația în cadrul unui grup este mai redusă, cu atât crește omogenitatea grupului respectiv.

Astfel, punctele se atribuie unui grup astfel încât suma pătratelor distanțelor dintre elementele grupului să fie minimă. În practică însă, este inefficient calculul similarității dintre fiecare două elemente ale setului de date, de aceea problema se reformulează astfel: pentru fiecare grup se determină și se actualizează un centru (punct central) și se atribuie elementele grupului cu centrul cel mai apropiat (cu similaritate maximă).

Metoda presupune mai întâi stabilirea *a priori* (de la început) a numărului de grupuri K. Apoi, pentru fiecare element, se determină grupul cu centrul cel mai apropiat și se consideră că elementul aparține acestui grup. După ce se prelucrează toate elementele în această manieră, centrul fiecărui cluster se recalculează ca fiind *baricentrul* sistemului format din punctele identificate anterior ca aparținând clusterului. Baricentrul se determină în poziția ce minimizează distanțele până la elementele grupului. Odată recalculate pozițiile tuturor centrelor grupurilor, se determină din nou distanțele până la fiecare element și se regroupează elementele în mod corespunzător. Raționamentul se reia până când centrele grupurilor devin stabile (nu se mai modifică semnificativ). În Fig 1 se ilustrează un exemplu de grupare a unei mulțimi de puncte din plan.

Presupunem că se dorește gruparea unei mulțimi de elemente în  $K$  grupuri. Algoritmul implică următorii pași:

1. Se stabilesc cele  $K$  centre în spațiul elementelor ce trebuie grupate. Pozițiile centrelor pot fi, de exemplu, generate aleator.
2. Pentru fiecare element, se determină grupul căruia îi aparține ca fiind cel cu centrul cel mai apropiat.
3. Odată grupate toate elementele, se actualizează centrul fiecărui grup (de exemplu, în cazul unor puncte din plan se poate face media aritmetică a coordonatelor).
4. Se repetă pașii 2, 3 până când pozițiile centrelor grupurilor nu se mai modifică (în practică, se stabilește o valoare prag sub care trebuie să se situeze diferența dintre pozițiile centrelor grupurilor determinate în cadrul iterației curente și cele determinate în iterația anterioară).

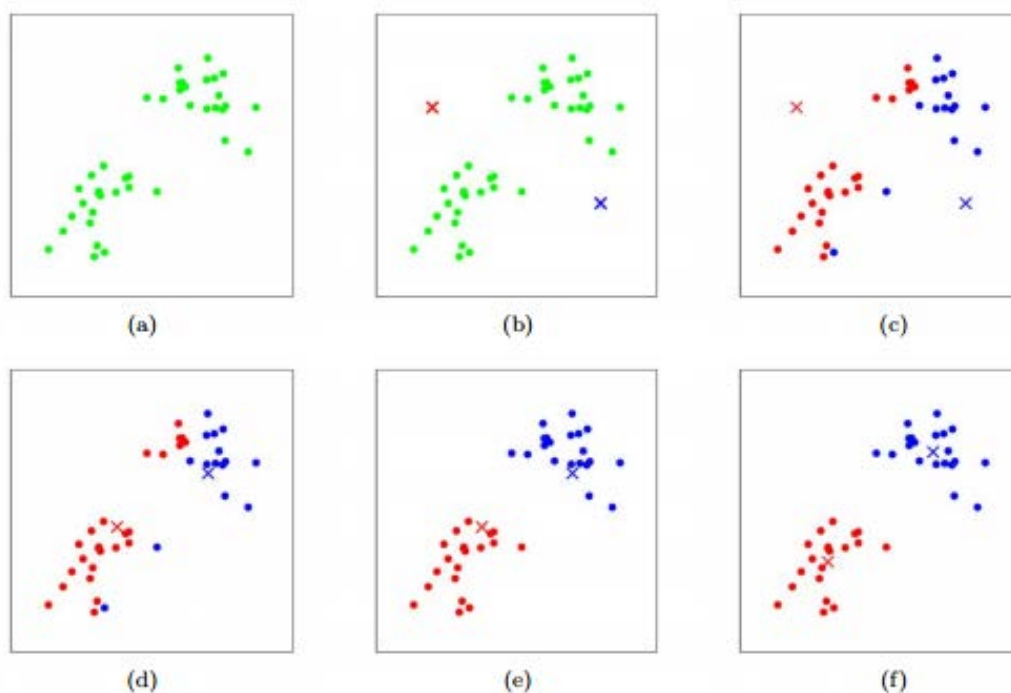


Fig. 1. Exemplu de grupare K-means: (a) Punctele în configurația inițială, negrupate; (b) Se stabilește  $K=2$  și se inițializează două centre, marcate cu X, în poziții aleatoare; (c) Se realizează o primă grupare: fiecare punct aparține grupului cu centrul cel mai apropiat; (d, e, f) în cadrul mai multor iterații se recalculează pozițiile centrelor (media aritmetică a pozițiilor punctelor) și se regroupează punctele folosind noile centre.

**Cerințe:**

1. Implementați algoritmul K-means pentru o mulțime de puncte din plan și reprezentați rezultatul grupării (Fig. 2). Se poate utiliza ca punct de plecare codul sursă care însoțește documentația laboratorului. Testați implementarea folosind punctele din fișierele knnpoints3.txt, knnpoints4.txt, knnpoints\_uniform.txt

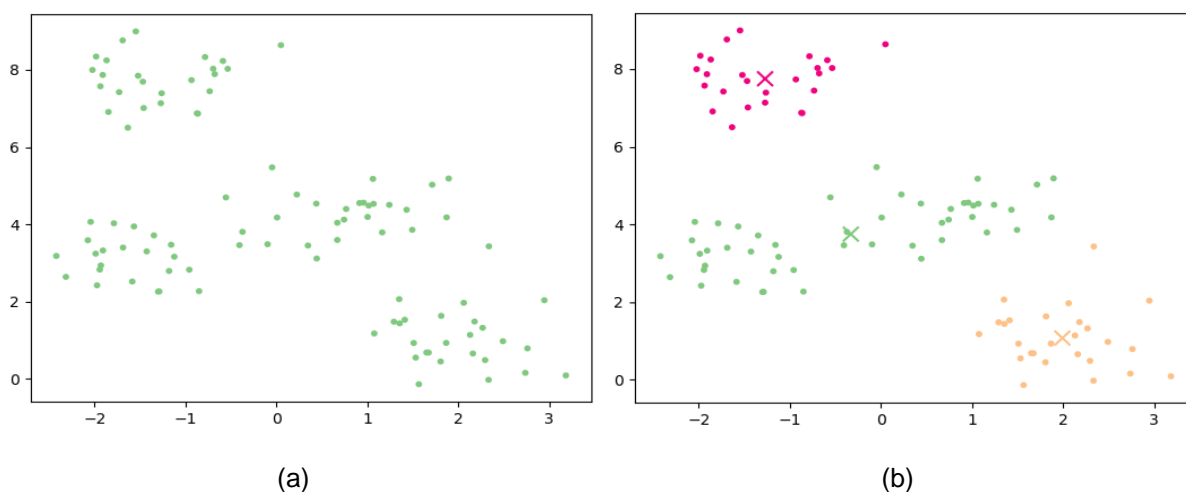


Fig. 2. (a) Punctele inițiale, negrupate; (b) Punctele grupate în trei grupuri (K=3). Centrele grupurilor sunt marcate cu X

2. Determinați cea mai bună valoare a lui K (cea pentru care grupurile sunt cât mai compacte și mai bine separate) folosind metoda coeficientului siluetă.

Coeficientul siluetă (*Silhouette Coefficient*, notat SC) se calculează astfel:

Pentru fiecare punct  $i$  se determină:

$a(i)$  = distanța medie dintre  $i$  și celelalte puncte din grupul din care face parte  $i$

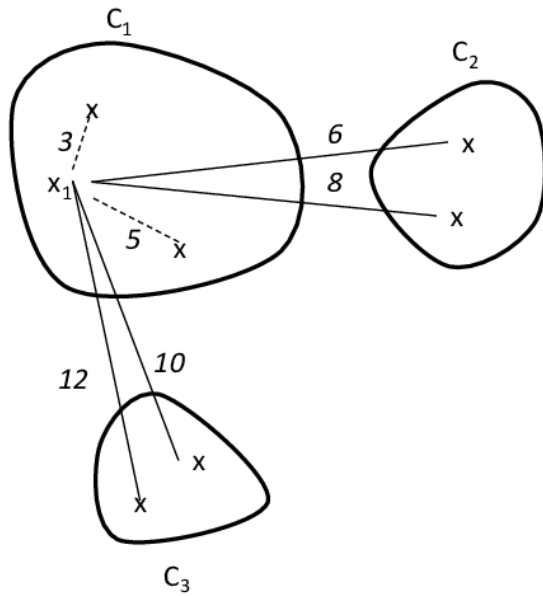
$b(i)$  = minimul distanțelor medii dintre  $i$  și punctele din celelalte grupuri

$s(i) = (b(i) - a(i)) / \max(a(i), b(i))$        $s(i)$  – silueta punctului  $i$

În Fig. 3. se prezintă un exemplu de calcul al siluetei unui punct.

$SC = s(i)$  mediu (silueta medie determinată pentru toate punctele, pentru o anumită valoare a lui K)

Cea mai bună valoare a lui K = cea pentru care SC este maxim.



$$a(x_1) = \frac{3 + 5}{2} = 4$$

$$b(x_1) = \min\left(\frac{6+8}{2}, \frac{10+12}{2}\right) = 7$$

$$s(x_1) = \frac{7-4}{\max(7,4)} = \frac{3}{7}$$

Fig. 3. Exemplu de calcul al siluetei  $s(x_1)$  a punctului  $x_1$ , pentru  $K = 3$

## Clasificarea datelor. K-Nearest Neighbors

Clasificarea datelor constituite dintr-o mulțime de elemente de un anumit tip constă în stabilirea categoriei în care se încadrează elementele pe baza unor trăsături, caracteristici sau atribute ale acestora. Categoriile în care trebuie încadrate elementele se numesc *clase* și trebuie să fie în număr finit.

Algoritmii care realizează clasificarea se numesc *clasificatori* – scopul lor este de a stabili o clasă (dintr-o mulțime finită de clase) pentru oricare element dintr-o mulțime sau un domeniu, realizând diverse prelucrări pe baza trăsăturilor acelor elemente.

Clasificarea este un proces de învățare supervizată, ceea ce înseamnă că un clasificator necesită o mulțime finită de elemente deja clasificate (*mulțimea datelor de antrenare*), pe baza cărora învață să clasifice și alte elemente de același tip. Exemplu: Un clasificator care realizează recunoașterea unor persoane din imagini necesită o mulțime de imagini pre-clasificate, unde persoanele să fie deja cunoscute. Pe baza acestor imagini, clasificatorul va învăța să recunoască aceleași persoane în (teoretic) orice altă imagine de același tip.

**K-Nearest Neighbors (KNN)** este o metodă de clasificare în cadrul căreia, pentru a se stabili clasa de apartenență a unui element, se iau în calcul cei mai apropiați  $k$  vecini cunoscuți ai acelui element. Datele de antrenare constau într-o mulțime de elemente deja clasificate. La apariția unui nou element, clasa acestuia se decide funcție de clasele celor mai apropiați  $k$  vecini cunoscuți (în cel mai simplu caz, noul element este încadrat în clasa majoritară prezentă printre cei  $k$  vecini ai săi). Pentru a se stabili cât de apropiat este un vecin, este nevoie de o metrică de similaritate prin intermediul căreia să se determine “distanța” dintre oricare două elemente, similar cu abordarea de la laboratorul anterior.

### Exemplu:

Dorim să clasificăm o serie de elemente care aparțin mulțimii punctelor din plan. Pentru aceasta, stabilim de la bun început următoarele aspecte:

- clasele în care vom încadra punctele: în cazul de față presupunem două clase, denumite **roșu** și **verde**.
- trăsăturile elementelor (caracteristicile elementelor pe baza cărora se va realiza clasificarea): în cazul punctelor din plan, aceste caracteristici vor fi coordonatele  $x, y$
- metrica de similaritate: o modalitate de determinare a distanței dintre două puncte, pe baza trăsăturilor menționate anterior. În cazul de față, vom folosi distanța euclidiană (Fig. 1)

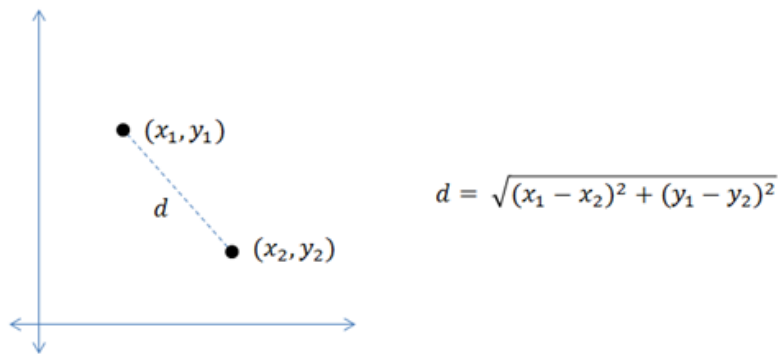


Fig 1. Distanța euclidiană dintre două puncte din plan

- o mulțime de date de antrenare: o serie de puncte care sunt deja încadrate într-una din cele două clase. Pe baza acestor puncte, algoritmul de clasificare va putea generaliza – va putea stabili clasa în care se încadrează oricare punct din plan

Principiul de bază al **KNN** constă în faptul că apartenența unui punct la o clasă se determină pornind de la cei mai apropiați  $k$  vecini din mulțimea de antrenare.

Fie mulțimea de puncte din Fig. 2. Acestea sunt deja încadrate în clasele **roșu** sau **verde**, scopul fiind să determinăm clasa oricărui alt punct.

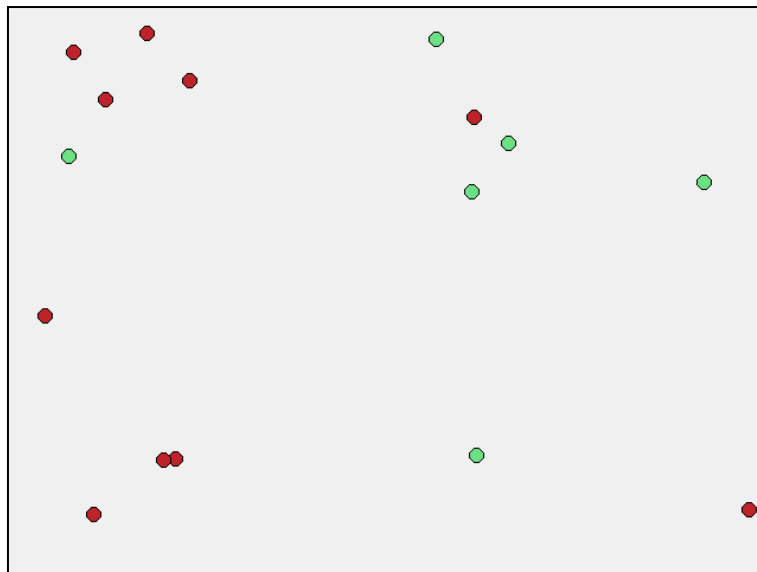


Fig. 2. Un set de puncte împărțite în două categorii. Clasele aferente sunt marcate prin culori diferite.

Stabilim o valoare a paramerului  $k = 3$ . Pentru oricare nou punct, clasa în care se încadrează se decide funcție de clasele celor mai apropiate trei puncte dintre cele deja cunoscute. Spre exemplu, dacă dintre cele mai apropiate trei puncte, două aparțin clasei **roșu** iar al treilea aparține clasei **verde**, putem decide că noul punct aparține clasei **roșu**, deoarece aceasta este clasa majorității vecinilor săi (Fig. 3(a)). Un alt exemplu este ilustrat în Fig. 3(b), unde noul punct aparține clasei **verde**, cea a tuturor vecinilor săi.

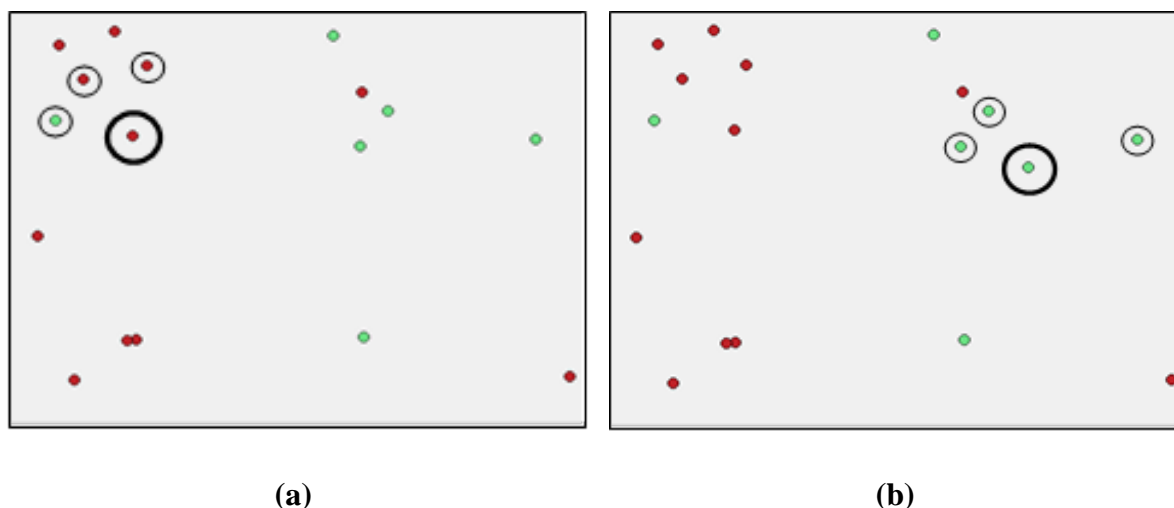


Fig. 3. Două exemple de clasificare a unor noi puncte. Acestea vor aparține clasei majoritare a celor mai apropiați  $k=3$  vecini

### K-nearest neighbor cu ponderi

Până acum, toți vecinii noului punct au contribuit în mod echitabil la stabilirea clasei acestuia, ei contând doar ca număr. Se pot însă defini ponderi pentru fiecare vecin, acestea semnificând importanța vecinilor în procesul de stabilire a clasei noului punct.

Cel mai frecvent se consideră ca pondere inversul pătratului distanței vecinului față de punct. Așadar, cu cât un vecin este mai apropiat de punct, cu atât crește probabilitatea ca punctul să aparțină clasei aceluia vecin.

Fața de exemplele anterioare, vecinii nu conteaza doar ca număr, ci decizia se ia funcție de suma ponderilor vecinilor din fiecare clasă. Noul punct va fi asignat clasei cu suma cea mai mare.



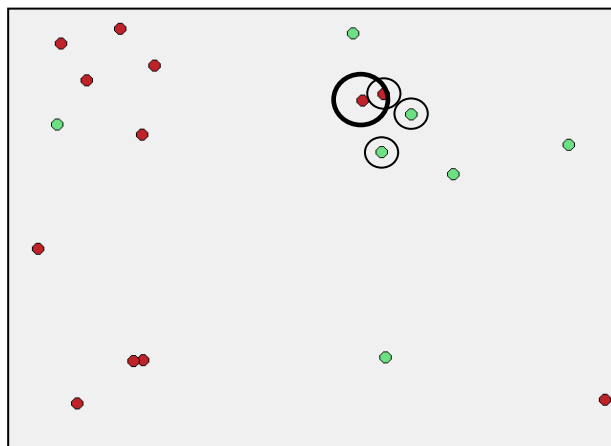


Fig. 4. Punct clasificat ținând cont de ponderile vecinilor. Vecinul **roșu** este mult mai apropiat de noul punct decât ceilalți doi vecini, așadar ponderea sa (inversul patratului distanței) este mai mare decât suma ponderilor celorlalți doi vecini.

În Fig. 4, pentru  $k = 3$ , se observă că, deși vecinii majoritari ai punctului nou adăugat aparțin clasei **verde**, punctul vizat se va încadra în clasa corespunzătoare vecinului **roșu**. Acesta, fiind mult mai apropiat decât ceilalți doi, are o pondere mult mai mare. Principul este explicat în detaliu în Tabelul 1.

Tabelul 1. Clasificarea folosind KNN cu și fără ponderi

KNN fără ponderi, K=3	KNN cu ponderi, K=3
<ul style="list-style-type: none"> <li>- clasa predominantă se determină contorizând vecinii</li> <li>- dintre cei mai apropiați K=3 vecini: <ul style="list-style-type: none"> <li>- doi aparțin clasei <b>roșu</b></li> <li>- unul aparține clasei <b>verde</b></li> </ul> </li> <li>- astfel, un nou punct X va fi asignat clasei <b>roșu</b></li> </ul>	<ul style="list-style-type: none"> <li>- clasa predominantă se determină efectuând suma ponderilor vecinilor:</li> <li>- <math>W_{\text{red}} = W_2 + W_3 = 1 / \text{dist}(X, P_2)^2 + 1 / \text{dist}(X, P_3)^2</math></li> <li>- <math>W_{\text{green}} = W_1 = 1 / \text{dist}(X, P_1)^2</math></li> <li>- dacă <math>W_{\text{green}} &gt; W_{\text{red}}</math> atunci lui X I se atribuie clasa <b>verde</b></li> </ul>

## Testarea clasificatorului

Testarea presupune verificarea acurateții clasificatorului antrenat. Pentru aceasta, se stabilește o mulțime de puncte de test, a căror clasă se cunoaște deja, dar care nu sunt implicate în procesul de antrenare. Se aplică algoritmul de clasificare pe aceste puncte și se determină, procentual, câte puncte sunt clasificate greșit de către algoritmul de clasificare (rezultă *eroarea de clasificare* a clasificatorului)

**Exemplu:** presupunem ca mulțimea inițială este formată din 100 puncte. Dintre acestea:

- 60 de puncte se vor folosi pentru antrenare (vor constitui mulțimea de antrenare)
- 40 de puncte se vor folosi pentru testare (vor constitui mulțimea de test)

Aplicăm KNN pe cele 40 puncte de test (individual) folosind vecini preluați dintre punctele din mulțimea de antrenare. Interesează câte dintre cele 40 puncte de test sunt clasificate greșit (pentru câte dintre punctele de test clasa determinată de KNN va fi alta decât cea inițială a punctului). Atunci eroarea de clasificare va fi:

$$err = \frac{nr\_puncte\_clasificate\_gresit}{nr\_total\_de\_puncte\_test}$$

## Validarea clasificatorului

Validarea se referă la ajustarea parametrilor clasificatorului astfel încât să se minimizeze eroarea de clasificare. În cazul de față, singurul parametru al KNN este K, numărul de vecini. Scopul este de a determina cea mai bună valoare a lui K, cea pentru care eroarea de clasificare este minimă. În general, pentru validare se creează o mulțime separată a punctelor, mulțimea de validare.

## Cerințe

1) Implementați algoritmul K-nearest neighbor, variantele fără ponderi, respectiv cu ponderi. Pentru datele furnizate odată cu documentația. La apăsarea butonului stâng al mouse-ului se va adăuga un nou punct pentru care se va determina clasa de apartenență funcție de cei mai apropiați  $k$  vecini. Ca punct de plecare, se poate utiliza codul pus la dispoziție pe lângă documentație.

La apăsarea butonului mouse-ului:

- Se adaugă un nou punct
- Se determină distanțele față de toate punctele existente
- Se sortează ascendent punctele funcție de distanțe
- Se consideră primele  $k$  puncte dintre cele sortate (acestea vor fi cei mai apropiați  $k$  vecini)
- Se determină clasa punctului adăugat
  - o Dacă nu se iau în calcul ponderile, se determină histograma vecinilor pe clase (adică se determină numărul vecinilor care aparțin fiecărei clase), iar clasa corespunzătoare punctului adăugat va fi cea cu numărul maxim de vecini
  - o Dacă se iau în calcul ponderile, acestea se determină pentru fiecare vecin ca fiind  $1/d^p$ , unde  $d$  este distanța de la vecin până la punctul adăugat. Se sumează aceste ponderi pe clase, iar clasa punctului adăugat se consideră a fi cea cu suma maximă

2) Determinați eroarea de clasificare a KNN pentru datele inițiale, nemodificate (nu se iau în calcul alte puncte adăugate ulterior), pentru o anumită valoare a lui  $K$  (poate fi  $K=3$ )

Pentru aceasta, împărțiți datele disponibile astfel: 60% pentru antrenare și 40% pentru testare.

Aplicați KNN pe punctele din mulțimea de test preluând vecinii din mulțimea de antrenare.

Determinați eroarea de clasificare pentru ambele abordări ale algoritmului (cu/fără ponderi) ca fiind:

$$err = \frac{nr\_puncte\_clasificate\_gresit}{nr\_total\_de\_puncte\_test}$$

2) Realizați o validare a algoritmului pentru identificarea celei mai bune valori a lui  $K$ .

Împărțiți mulțimea punctelor în tre submulțimi astfel:

- 60% dintre puncte pentru antrenare
- 20% pentru validare
- 20% pentru test

Aplicați următoarele două metode:

a) *Validare simplă*:

Pentru fiecare valoare a lui  $k$ , de la 1 la o valoare maximă  $k_{\max}$ , se determină eroarea de clasificare pentru punctele din mulțimea de validare folosind vecinii preluați din mulțimea de antrenare. Cea mai bună valoare a lui  $k$  este cea pentru care eroarea este minimă. Testați algoritmul cu această valoare a lui  $k$  folosind mulțimea de test.

b) *Validare încrucișată (Cross validation)*:

Se împarte mulțimea datelor într-o mulțime de antrenare și una de test.

Pentru cea de antrenare:

Se formează  $n$  subgrupuri egale folosind punctele din setul de date. Așadar, pentru fiecare  $i = 1..n$ , datele se vor împărți astfel:

- un subgrup  $i$  de test
- mulțimea formată din restul de  $n-1$  subgrupuri, cea care se va folosi efectiv pentru antrenare

O astfel de partiționare se numește *split*, iar grupul  $i$  se mai numește *fold*.

Modalitatea de divizare a mulțimii punctelor se ilustrează în Fig. 5.

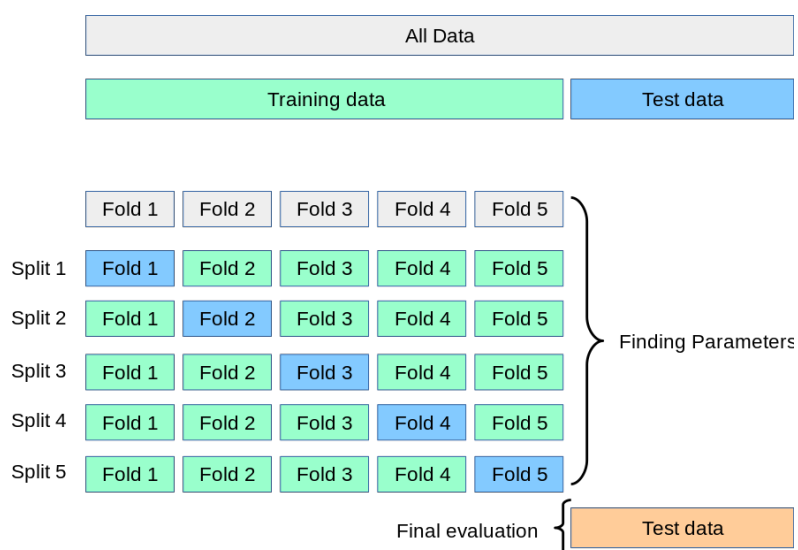


Fig. 5. Modalitatea de partiționare a mulțimii datelor pentru realizarea validării încrucișate și a testării.

- pentru fiecare subgrup  $i$ ,  $i = 1..n$  se obține câte o eroare pentru fiecare valoare a lui  $k$ ,  $k=1..k_{\max}$ .
- astfel, pentru fiecare valoare a lui  $k$  vom obține  $n$  erori de clasificare, care rezultă din validarea efectuată pentru fiecare din cele  $n$  subgrupuri.
- pentru fiecare valoare a lui  $k$  se determină eroarea medie.

Valoarea finală a lui  $k$  este cea cu eroarea medie minimă. În final, se testează clasificatorul cu această valoare a lui  $k$  folosind mulțimea de test.

## Naive Bayes

Naive Bayes este metodă de clasificare ce are la bază teorema elaborată de Thomas Bayes (Eq. 1). În caz general, presupunem că datele de intrare se reprezintă printr-un vector de caracteristici (*feature vector*) notat  $x = (x_1, \dots, x_n)$ . Acest vector conține o serie de valori pentru  $n$  atribute specifice fiecărei instanțe din setul de date (altfel spus, conține valorile caracteristicilor fiecărui element din mulțimea de obiecte inițială). Pentru obiectele dintr-o mulțime de date de un anumit tip se folosește denumirea mult mai generală de *instanță*. Presupunem că instanțele se încadrează în  $K$  clase, notate  $c_1 \dots c_K$ . Pentru o nouă instanță, neclasificată, dorim să determinăm probabilitățile de încadrare a acesteia în cele  $K$  clase, pe baza analizei caracteristicilor instanțelor cărora li se cunosc deja clasele.

$$P(c_k | X) = \frac{P(X | c_k) P(c_k)}{P(X)}, k = 1..K \quad (1)$$

Eq. 1 se interpretează astfel:

$P(c_k | X)$  este probabilitatea ca o nouă instanță cu atributele date de vectorul  $X$  să aparțină clasei  $c_k$  (Este ceea ce dorim să determinăm.)

$P(X | c_k)$  este probabilitatea ca în clasa  $c_k$  să existe o instanță cu vectorul de atribute  $X$

$P(c_k)$  este probabilitatea ca o instanță să se încadreze în clasa  $c_k$

$P(X)$  este probabilitatea de apariție a unei instanțe cu vectorul de atribute  $X$

În cazul (cel mai frecvent întâlnit) în care instanțele au mai multe atribute, probabilitatea de apariție a unei instanțe cu un vector de valori ale acestor atribute în clasa oarecare  $c_k$  se calculează conform Eq. 2.

$$P(X | c_k) = \prod_{i=1}^n P(x_i | c_k) \quad (2)$$

Interpretarea Eq. 2 este următoarea: presupunând că atributele sunt independente unele de altele, probabilitatea ca în clasa  $c_k$  să apară o instanță cu vectorul  $X$  cu  $n$  atribute se calculează ca fiind produsul probabilităților ca în clasa  $c_k$  să apară o instanță cu valoarea  $x_i$  a atributului  $i$ .

Odată determinate probabilitățile de încadrare a noii instanțe în cele  $c_k$  clase, se consideră că aceasta aparține clasei pentru care s-a obținut probabilitatea maximă.

Având în vedere faptul că termenul  $P(X)$  din Eq. 1 este același pentru toate clasele  $c_k$ , el poate fi ignorat în contextul în care trebuie maximizat termenul din partea dreaptă a Eq. 1. Luând în considerare acest fapt, precum și Eq. 2, calculul probabilităților se poate simplifica (Eq. 3).

$$P(c_k | X) = P(c_k) \prod_{i=1}^n P(x_i | c_k) \quad (3)$$

### Exemplu:

Considerăm următorul set de date:

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
1	Soare	Mare	Mare	Absent	Nu
2	Soare	Mare	Mare	Prezent	Nu
3	Înnorat	Mare	Mare	Absent	Da
4	Ploaie	Medie	Mare	Absent	Da
5	Ploaie	Mică	Normală	Absent	Da
6	Ploaie	Mică	Normală	Prezent	Nu
7	Înnorat	Mică	Normală	Prezent	Da
8	Soare	Medie	Mare	Absent	Nu
9	Soare	Mică	Normală	Absent	Da
10	Ploaie	Medie	Normală	Absent	Da
11	Soare	Medie	Normală	Prezent	Da
12	Înnorat	Medie	Mare	Prezent	Da
13	Înnorat	Mare	Normală	Absent	Da
14	Ploaie	Medie	Mare	Prezent	Nu

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
15	Soare	Mare	Normală	Absent	???

Problema care se pune e legată de oportunitatea de a practica un sport oarecare (“Joc”), având în vedere condițiile date de starea vremii, temperatură, umiditate și vânt. Setul de date cu care se pornește este dat de instanțele 1..14, pentru care se cunoaște clasa (sunt două clase, Da și Nu). Fiecare instanță constă în vectorul de valori ale celor 4 atribute.

Fiind dată o nouă instanță, a 15-a, în ce clasă se încadrează aceasta? Altfel spus, se dorește sau nu practicarea sportului în condițiile meteorologice definite de valorile atributelor instanței 15, având în vedere deciziile care s-au luat anterior, în cazul celorlalte 14 instanțe?

Trebuie să calculăm probabilitățile ca instanța 15 să se încadreze în cele două clase Da, Nu, și să alegem clasa corespunzătoare probabilității maxime.

- Calculăm probabilitatea de apariție a clasei  $Da$ , notată  $J_D$ . Observăm că ea apare de 9 ori, din cele 14 cazuri posibile. Prin urmare,

$$P(J_D) = \frac{9}{14}$$

- Calculăm probabilitățile condiționate de apartenența la clase (termenii produsului din Eq. 3). Pentru atributul *Starea vremii*, observăm că 2 instanțe din cele 9 din clasa  $Da$  au valoarea *Soare* (cea care provine din instanța 15, nou apărută), notată  $S_s$ . Așadar,

$$P(S_s | J_D) = \frac{2}{9}$$

În mod similar, determinăm probabilitățile valorilor atributelor instanței 15, condiționate de apartenența la clasa  $Da$ .

$$P(T_H | J_D) = \frac{2}{9}$$

$$P(U_N | J_D) = \frac{6}{9}$$

$$P(V_A | J_D) = \frac{6}{9}$$

Calculăm probabilitatea ca noua instanță să aparțină clasei  $Da$ , conform cu Eq. 3:

$$P(J_D) \cdot P(x_i | J_D) = \frac{9}{14} \cdot \frac{2}{9} \cdot \frac{2}{9} \cdot \frac{6}{9} \cdot \frac{6}{9} = 14.109 \cdot 10^{-3}$$

Efectuând aceleași calcule pentru clasa  $Nu$ , obținem:

$$P(J_N) \cdot P(x_i | J_N) = \frac{5}{14} \cdot \frac{3}{5} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{2}{5} = 6.857 \cdot 10^{-3}$$

Valoarea maximă este cea corespunzătoare clasei  $Da$ , în care se va încadra noua instanță.

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
15	Soare	Mare	Normală	Absent	<b>Da</b>



## Corecția Laplace

Există posibilitatea ca, pentru anumite seturi de date, valorile unora dintre termenii produselor din Eq. 3 să fie nule, caz în care produsul corespunzător va fi, la rândul său, nul.

### Exemplu:

Fie următorul set de date:

Starea vremii	Umiditate	Joc
Înnorat	Mare	Da
Ploaie	Mare	Da
Înnorat	Mare	Da
Soare	Mare	Nu
Soare	Mare	Nu
Ploaie	Normală	Nu

Dorim să clasificăm următoarea instanță:

Starea vremii	Umiditate	Joc
Înnorat	Normală	Nu

Realizând aceleași calcule ca și în cazul anterior, obținem:

$$P(J_D) \cdot P(x_i|J_D) = \frac{3}{6} \cdot \frac{2}{3} \cdot \frac{0}{3} = 0,$$

$$P(J_N) \cdot P(x_i|J_N) = \frac{3}{6} \cdot \frac{0}{3} \cdot \frac{1}{3} = 0.$$

Pentru a preîntâmpina această situație, considerăm modul în care se calculează probabilitatea de apariție a atributului  $x_i$  printre valorile instanțelor din clasa  $c_j$  (Eq. 4), unde  $n_{ij}$  este numărul de apariții a unei valori a atributului  $i$  în clasa  $j$ , iar  $n_j$  este numărul de apariții al clasei  $j$ .

$$P(x_i|c_j) = \frac{n_{ij}}{n_j} \quad (4)$$

Corecția Laplace se aplică conform cu Eq. 5, unde  $c$  este numărul de clase:

$$P(x_i|c_j) = \frac{n_{ij} + 1}{n_j + c} \quad (5)$$

Această abordare garantează faptul că termenii produsului vor fi nenuli. În cazul exemplului anterior, calculele vor fi:

$$P(J_D) \cdot P(x_i|J_D) = \frac{3}{6} \cdot \frac{2+1}{3+2} \cdot \frac{0+1}{3+2} = 0.06$$

$$P(J_N) \cdot P(x_i|J_N) = \frac{3}{6} \cdot \frac{0+1}{3+2} \cdot \frac{1+1}{3+2} = 0.04$$

Prin urmare, se poate decide faptul că noua instanță se va încadra în clasa *Da*.

În cazul exemplului inițial, vom avea:

$$P(J_D) \cdot P(x_i|J_D) = \frac{9}{14} \cdot \frac{2+1}{9+2} \cdot \frac{2+1}{9+2} \cdot \frac{6+1}{9+2} \cdot \frac{6+1}{9+2} = 19.363 \cdot 10^{-3}$$

$$P(J_N) \cdot P(x_i|J_N) = \frac{5}{14} \cdot \frac{3+1}{5+2} \cdot \frac{2+1}{5+2} \cdot \frac{1+1}{5+2} \cdot \frac{2+1}{5+2} = 10.71 \cdot 10^{-3}.$$

Rezultatul clasificării nu se schimbă, deoarece contează doar comparația, nu și valorile propriu-zise.

### Cerințe:

- 1) Implementați metoda de clasificare Naive Bayes folosind setul de date din fișierul *data\_vreme1.csv*:

Starea vremii	Temperatura	Umiditate	Vant	Joc
Soare	Mare	Mare	Absent	Nu
Soare	Mare	Mare	Prezent	Nu
Innorat	Mare	Mare	Absent	Da
Ploaie	Medie	Mare	Absent	Da
Ploaie	Mica	Normala	Absent	Da
Ploaie	Mica	Normala	Prezent	Nu
Innorat	Mica	Normala	Prezent	Da
Soare	Medie	Mare	Absent	Nu
Soare	Mica	Normala	Absent	Da
Ploaie	Medie	Normala	Absent	Da
Soare	Medie	Normala	Prezent	Da
Innorat	Medie	Mare	Prezent	Da
Innorat	Mare	Normala	Absent	Da
Ploaie	Medie	Mare	Prezent	Nu

Programul trebuie să permită clasificarea unor instanțe precizate de utilizator. Se vor afișa probabilitățile ambelor clase și se va evidenția clasa cu probabilitate maximă, adică decizia de clasificare. Utilizatorul va avea posibilitatea să utilizeze sau nu corecția Laplace.

Pentru verificare, classificați instanța 15 menționată anterior.

- 2) Implementați metoda de clasificare Naive Bayes folosind setul de date din fișierul *data\_vreme2.csv*:

Starea vremii	Temperatura	Umiditate	Vant	Joc
Soare	17	Mare	Absent	Nu
Soare	15	Mare	Prezent	Nu
Innorat	24	Mare	Absent	Da
Ploaie	19	Mare	Absent	Da
Ploaie	19	Normala	Absent	Da
Ploaie	12	Normala	Prezent	Nu
Innorat	20	Normala	Prezent	Da
Soare	18	Mare	Absent	Nu
Soare	18	Normala	Absent	Da
Ploaie	17	Normala	Absent	Da
Soare	19	Normala	Prezent	Da
Innorat	15	Mare	Prezent	Da
Innorat	20	Normala	Absent	Da
Ploaie	18	Mare	Prezent	Nu

De data aceasta, atributul **Temperatura** are valori numerice, a căror prelucrare necesită o altă abordare: pentru valorile acestui atribut, vom determina probabilitățile folosind funcția densitate de probabilitate a distribuției acestora.

Pentru fiecare clasă (Da, Nu) valorile temperaturii sunt generate conform cu distribuția normală. Așadar pentru aceste valori se poate obține probabilitatea în raport cu clasa corespunzătoare folosind funcția densitate de probabilitate a distribuției normale, cu următoarea expresie:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (6)$$

unde  $\mu$  ("miu") și  $\sigma$  ("sigma") sunt *media*, respectiv *abaterea standard* a distribuției. Aceste valori trebuie determinate separat pentru fiecare clasă, din valorile cunoscute ale temperaturii.

Media  $\mu$  se determină ca fiind media aritmetică a valorilor. Presupunând  $n$  valori ale temperaturii,  $t_1 \dots t_n$ , formula este următoarea:

$$\mu = \frac{1}{n} \sum_{i=1}^n t_i \quad (7)$$

Abaterea standard  $\sigma$  se determină astfel:

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (t_i - \mu)^2} \quad (8)$$

**Pentru verificare:**

Pentru datele din fișierul *data\_vreme2.csv*, rezultă următoarele valori:

$$\mu_{DA} = 19$$

$$\sigma_{DA} = 2.3094$$

$$\mu_{NU} = 16$$

$$\sigma_{NU} = 2.28035$$

Pentru următoarea instanță:

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
15	Soare	14	Normală	Absent	???

Probabilitatea valorii temperaturii rezultă, pentru cele două clase, din funcția densitate de probabilitate (Eq. 6) cu parametri  $\mu$  și  $\sigma$  determinați pentru fiecare clasă:

$$P(\text{Temp} = 14 \mid Da) = 0.0166$$

$$P(\text{Temp} = 14 \mid Nu) = 0.1190$$

Calculăm probabilitatea ca noua instanță să aparțină clasei *Da*, conform cu Eq. 3:

$$P(J_D) \cdot P(x_i \mid J_D) = 1.05 \cdot 10^{-3}$$

Efectuând aceleași calcule pentru clasa *Nu*, obținem:

$$P(J_N) \cdot P(x_i \mid J_N) = 2.04 \cdot 10^{-3}$$

Așadar, instanța anterioară se clasifică astfel:

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
15	Soare	14	Normală	Absent	<b>Nu</b>

## Arbori de decizie ID3

Arborii de decizie sunt metode de clasificare ce presupun organizarea ierarhica a atributelor (caracteristicilor) unor date dintr-un anumit domeniu, la baza ierarhiei aflându-se posibilele decizii (clasele) care rezultă din parcurgerea acelor atribute. Arborii ID3 (Iterative Dichotomizer) ordonează ierarhic caracteristicile datelor în funcție de câștigul informațional pe care îl conferă luarea unei decizii în baza acelor caracteristici.

Arborele are următoarea structură:

- Rădăcina și nodurile intermediare conțin atributele, în ordinea descendentă a importanței lor decizionale.
- Muchiile reprezintă valorile (sau grupurile de valori ale) atributelor.
- Fiecare nod are un număr de descendenți egal cu numărul deciziilor posibile pornind de la acel nod. În cel mai simplu caz, pentru atribute cu valori ordinale, un nod are câte un descendent pentru fiecare valoare a atributelor sale.
- Nodurile frunză conțin valorile claselor.

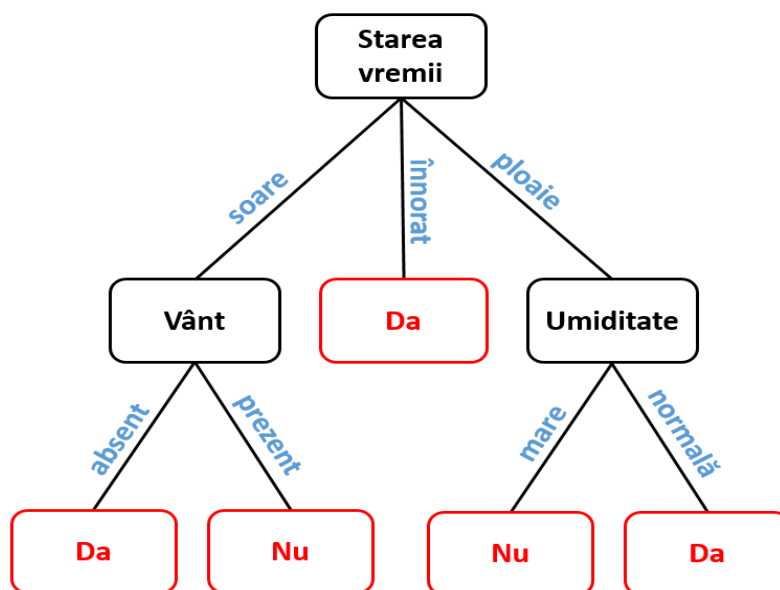
Clasificarea folosind arbori de decizie presupune:

- Generarea arborelui pornind de la un set de date de antrenare. Acest set de date conține o mulțime de instanțe ale căror clase se cunosc deja.
- Parcurgerea arborelui pentru o nouă instanță. Această parcurgere presupune divizarea repetată a soluțiilor posibile funcție de valorile atributelor instanțelor, până când se ajunge la un nod frunză care conține valoarea clasei ce i se atribuie instanței.

Fie următorul set de date de antrenare:

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
1	Ploaie	Mare	Mare	Absent	Nu
2	Ploaie	Mare	Mare	Prezent	Nu
3	Înnorat	Mare	Mare	Absent	Da
4	Soare	Medie	Mare	Absent	Da
5	Soare	Mică	Normală	Absent	Da
6	Soare	Mică	Normală	Prezent	Nu
7	Înnorat	Mică	Normală	Prezent	Da
8	Ploaie	Medie	Mare	Absent	Nu
9	Ploaie	Mică	Normală	Absent	Da
10	Soare	Medie	Normală	Absent	Da
11	Ploaie	Medie	Normală	Prezent	Da
12	Înnorat	Medie	Mare	Prezent	Da
13	Înnorat	Mare	Normală	Absent	Da
14	Soare	Medie	Mare	Prezent	Nu

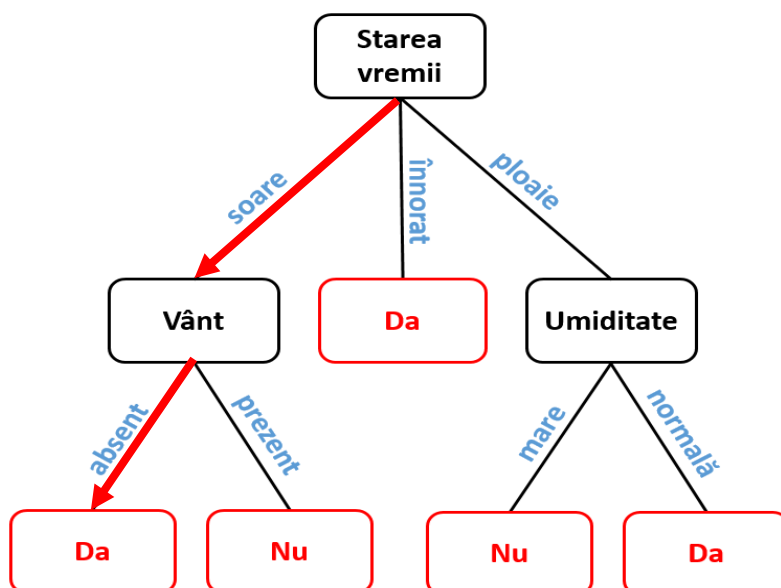
Arborele ID3 care se generează pe baza acestor date este:



Presupunem că dorim să stabilim clasa următoarei instanțe:

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
15	Soare	Mare	Normală	Absent	???

Pentru aceasta, parcurgem arborele în funcție de valorile atributelor instanței, până când întâlnim un nod frunză, care conține o clasă:



Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
15	Soare	Mare	Normală	Absent	Da

Pentru generarea arborelui trebuie stabilită o ordine a atributelor (care este atributul cel mai important, funcție de care se ia prima decizie? În exemplul de mai sus, de ce atributul din nodul rădăcină este “Starea vremii”?). Pentru a determina gradul de importanță, pentru fiecare atribut se calculează *câștigul informațional* (IG – Information Gain) care rezultă din luarea deciziei pornind de la acel atribut. Pentru arborele complet și pentru oricare subarbore al său, nodul rădăcină va conține atributul care conferă cel mai mare IG.

În cazul arborilor ID3, IG se determină ca fiind măsura în care se reduce *entropia* mulțimii datelor de antrenare în urma împărțirii acestora pe baza valorilor atributelor.

Pentru o mulțime de date  $S$ , entropia se calculează conform cu Eq. 1.

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x) \quad (1)$$

**unde:**

$X$  este mulțimea claselor - în exemplul de mai sus este mulțimea {Da, Nu}

$x$  este o clasă din mulțimea  $X$

$p(x)$  este probabilitatea de apariție a clasei  $x$  în setul de date  $S$

Entropia  $H(S)$  ia valori din  $[0, 1]$ . Valorile mari ale entropiei semnifică faptul că în  $S$  există un grad mare de incertitudine. Incertitudinea maximă înseamnă că cele două clase sunt prezente în mod egal în setul de date (50% din instanțe au clasa Da, 50% din instanțe au clasa Nu), caz în care  $H(S) = 1$ . La extrema cealaltă, toate clasele sunt fie Da, fie Nu, situație în care  $H(S) = 0$ . Câștigul informațional IG al unui atribut se referă la măsura în care scade entropia setului de date atunci când acesta este partajat de valorile acelui atribut.

De exemplu, atributul “Starea vremii” partiționează datele astfel:

Starea vremii = “Soare”:

Starea vremii	Temperatură	Umiditate	Vânt	Joc
Soare	Medie	Mare	Absent	Da
Soare	Mică	Normală	Absent	Da
Soare	Mică	Normală	Prezent	Nu
Soare	Medie	Normală	Absent	Da
Soare	Medie	Mare	Prezent	Nu

Starea vremii = “Ploaie”:

Starea vremii	Temperatură	Umiditate	Vânt	Joc
Ploaie	Mare	Mare	Absent	Nu
Ploaie	Mare	Mare	Prezent	Nu
Ploaie	Medie	Mare	Absent	Nu
Ploaie	Mică	Normală	Absent	Da
Ploaie	Medie	Normală	Prezent	Da

Starea vremii = “Înnorat”:

Starea vremii	Temperatură	Umiditate	Vânt	Joc
Înnorat	Mare	Mare	Absent	Da
Înnorat	Mică	Normală	Prezent	Da
Înnorat	Medie	Mare	Prezent	Da
Înnorat	Mare	Normală	Absent	Da

În urma acestei partiționări, entropia setului de date se determină ca fiind suma ponderată a entropiilor individuale ale partițiilor (Eq. 2).

$$H(S|A) = \sum_{t \in T} p(t)H(t) \quad (2)$$

**unde:**

$H(S|A)$  = entropia sistemului după partiționarea folosind valorile atributului A

$T$  = mulțimea partițiilor realizate folosind valorile atributului A

$t$  = o partiție din mulțimea T

$p(t)$  = probabilitatea valorii / valorilor atributului A care au generat partiția t

$H(t)$  = entropia partiției t

Câștigul informațional care rezultă în urma partiționării se determină ca fiind diferența dintre entropiile setului de date înainte și după partiționare (Eq. 3).

$$IG(S, A) = H(S) - H(S|A) \quad (3)$$

Pentru arborele de decizie și pentru oricare subarbore al său, dorim ca nodul rădăcină să corespundă cu atributul care oferă cel mai mare IG. Întrucât entropia întregului set de date  $H(S)$  este aceeași indiferent de modalitatea de partiționare, este suficient să determinăm atributul A pentru care  $H(S|A)$  este minimă. Astfel spus, *atributul din nodul rădăcină este cel care minimizează entropia indusă în setul de date în urma partiționării pe baza valorilor acelui atribut.*



Pentru setul de date din exemplu, calculăm entropia astfel (Eq. 1):

Joc	
Da	Nu
9	5

$$H(\text{Joc}) = - P_{(\text{Joc} = \text{Da})} \log_2(P_{(\text{Joc} = \text{Da})}) - P_{(\text{Joc} = \text{Nu})} \log_2(P_{(\text{Joc} = \text{Nu})}) = \\ = - 9/14 \log_2(9/14) - 5/14 \log_2(5/14) = \mathbf{0.94}$$

Pentru un atributul *Starea vremii*, calculul se face astfel (Eq. 2):

		Joc	
		Da	Nu
Starea vremii	Soare	3	2
	Înnorat	4	0
	Ploaie	2	3

$$H(\text{Joc}, \text{Starea vremii}) = P(\text{Soare}) * H(\text{Soare}) + P(\text{Înnorat}) * H(\text{Înnorat}) + P(\text{Ploaie}) * H(\text{Ploaie})$$

$$P(\text{Soare}) = 5/14$$

$$H(\text{Soare}) = - 3/5 \log_2(3/5) - 2/5 \log_2(2/5) = 0.971$$

$$P(\text{Înnorat}) = 4/14$$

$$H(\text{Înnorat}) = - 4/4 \log_2(4/4) - 0 = 0$$

$$P(\text{Ploaie}) = 5/14$$

$$H(\text{Ploaie}) = - 2/5 \log_2(2/5) - 3/5 \log_2(3/5) = 0.971$$

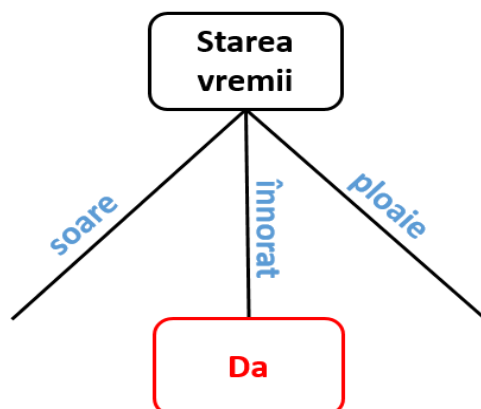
$$H(\text{Joc}, \text{Starea Vremii}) = 5/14 * 0.971 + 4/14 * 0 + 5/14 * 0.971 = \mathbf{0.693}$$

În mod similar, determinăm entropia celorlalte atribute și obținem:

		Joc	
		Da	Nu
Starea vremii	Soare	3	2
	Înnorat	4	0
	Ploaie	2	3
	<b>H = 0.693</b>		
Temperatură	Mare	2	2
	Medie	4	2
	Mică	3	1
	<b>H = 0.907</b>		
Umiditate	Mare	3	4
	Normală	6	1
	<b>H = 0.789</b>		
Vânt	Absent	6	2
	Prezent	3	3
	<b>H = 0.892</b>		

Observăm faptul că entropia minimă rezultă prin partiționarea folosind atributul *Starea vremii*, așadar prima decizie se va lua funcție de valorile acestuia (acest atribut va constitui rădăcina arborelui).

Pornind de la rădăcina stabilită anterior, se generează trei ramuri ce corespund celor trei valori ale atributului (Soare, Ploaie, Înnorat). Observăm că partiția corespunzătoare valorii *Înnorat* are entropia = 0, ceea ce este în acord cu faptul că toate instanțele din acea partiție au aceeași clasă (Joc = Da). Prin urmare, nodul cu care se încheie această ramură va fi o frunză cu valoarea clasei Da.

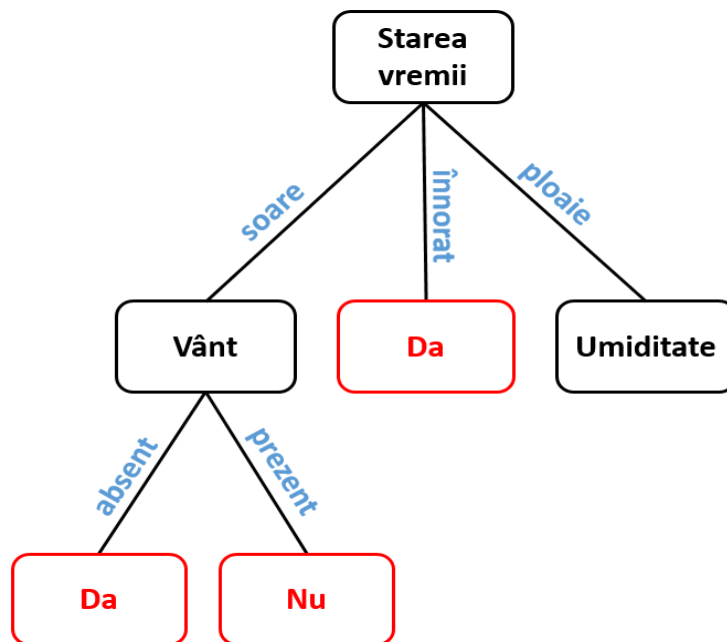


Celelalte ramuri au entropie nenulă, aşadar nodurile de la capetele lor vor fi intermediare (se vor subdiviza la rândul lor).

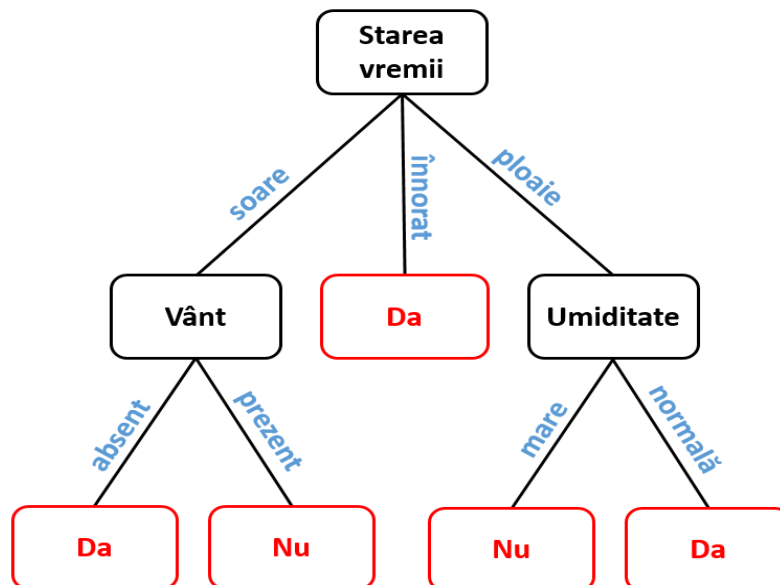
Pentru ramura *Soare*, setul de date devine:

Nr. instanță	Starea vremii	Temperatură	Umiditate	Vânt	Joc
1	Soare	Medie	Mare	Absent	Da
2	Soare	Mică	Normală	Absent	Da
3	Soare	Mică	Normală	Prezent	Nu
4	Soare	Medie	Normală	Absent	Da
5	Soare	Medie	Mare	Prezent	Nu

Efectuând aceleași calcule pentru cele trei atribute rămase (atributul *Starea vremii* a fost deja prelucrat), rezultă faptul că entropia minimă corespunde atributului *Vânt*. Entropia este nulă în cazul valorilor *Absent* și *Prezent*, ceea ce înseamnă că ramurile corespunzătoare lor se vor încheia cu noduri frunză. Acest lucru se observă și din tabel, unde toate instanțele cu *Vânt* = *Absent* se încadrează în clasa *Da*, iar cele cu *Vânt* = *Prezent* corespund clasei *Nu*.



În cazul ramurii Ploaie, se constată o situație asemănătoare pentru atributul Umiditate, iar subarboarele corespunzător are o structură similară. Rezultă astfel arborele complet:



**Cerințe:**

1) Generați un arbore de decizie ID3 pornind de la setul de date din fișierul *data\_vreme3.csv*:

Starea vremii	Temperatura	Umiditate	Vant	Joc
Ploaie	Mare	Mare	Absent	Nu
Ploaie	Mare	Mare	Prezent	Nu
Innorat	Mare	Mare	Absent	Da
Soare	Medie	Mare	Absent	Da
Soare	Mica	Normala	Absent	Da
Soare	Mica	Normala	Prezent	Nu
Innorat	Mica	Normala	Prezent	Da
Ploaie	Medie	Mare	Absent	Nu
Ploaie	Mica	Normala	Absent	Da
Soare	Medie	Normala	Absent	Da
Ploaie	Medie	Normala	Prezent	Da
Innorat	Medie	Mare	Prezent	Da
Innorat	Mare	Normala	Absent	Da
Soare	Medie	Mare	Prezent	Nu

O variantă de pseudocod pentru generarea arborelui:

ID3(Dataset, Lista\_Attribute)

- Generează un nod rădăcină
- Dacă toate instanțele din setul de date aparțin aceleiași clase, returnează un arbore cu un singur nod, rădăcina, cu valoarea clasei
- Dacă lista de attribute este goală, returnează un arbore cu un singur nod, rădăcina, cu valoarea clasei celei mai frecvente
- Determină atributul cu entropia minimă (notat A)
- Atributul nodului rădăcină este A
- Pentru fiecare valoare v a atributului A
  - o Adaugă o nouă ramură pentru rădăcină, corespunzătoare lui A = v
  - o Determină submulțimea Dataset(v), care conține instanțele cu valoarea v a atributului A
  - o Dacă Dataset(v):
    - Nu conține nici o instanță, atunci ramura se încheie cu un nod frunză, cu valoarea celei mai frecvente clase din Dataset
    - Conține instanțe, atunci ramura se continuă cu subarboarele ID3(Dataset(v), Lista\_Attribute - A)

Ca sugestie de implementare, se poate folosi codul sursă care însoțește laboratorul. La afișarea arborelui, se obține:

Starea vremii

```
--Innorat--> Da
--Ploaie--> Umiditate
--Normala--> Da
--Mare--> Nu
--Soare--> Vant
--Absent--> Da
--Prezent--> Nu
```

2) Folosiți arborele pentru a clasifica instanțele din fișierul *data\_vreme4.csv*. A clasifica o instanță înseamnă să parcurgem recursiv arborele pornind de la nodul rădăcină pe baza valorilor atributelor instanței, până când se ajunge într-un nod frunză ce conține valoarea unei clase.

Determinați eroarea de clasificare a arborelui pentru aceste instanțe.

Eroare de clasificare = nr. Instanțelor clasificate greșit / nr. tuturor instanțelor

Setul de date inițial, în care clasele sunt deja cunoscute:

Starea vremii	Temperatura	Umiditate	Vant	Joc
Soare	Mare	Normala	Absent	Da
Soare	Mare	Normala	Prezent	Nu
Ploaie	Medie	Mare	Prezent	Nu
Ploaie	Mare	Normala	Absent	Nu
Innorat	Mica	Mare	Prezent	Da

Folosind arborele ID3 generat anterior, aplicat individual pe fiecare instanță, rezultă clasele din ultima coloană:

Starea vremii	Temperatura	Umiditate	Vant	Joc	Joc – ID3
Soare	Mare	Normala	Absent	Da	Da
Soare	Mare	Normala	Prezent	Nu	Nu
Ploaie	Medie	Mare	Prezent	Nu	Nu
Ploaie	Mare	Normala	Absent	Nu	Da
Innorat	Mica	Mare	Prezent	Da	Da

Observăm faptul că, din cele 5 instanțe, una este clasificată greșit de către arborele ID3. Prin urmare, eroarea de clasificare este  $1/5 = 0.2$

## Clasificare bazată pe ansambluri. Random Forest

Random Forest este o metodă de învățare bazată pe ansambluri ce operează cu noțiunea de arbore de decizie, studiată în laboratorul anterior. Metodele bazate pe ansambluri funcționează pe principiul conform căruia un grup de „clasificatori slabi” (*weak classifiers*) pot forma, împreună, un clasificator puternic (*strong classifier*). Astfel, rezultatul clasificării folosind un ansamblu se determină prin compunerea rezultatelor individuale ale mai multor modele de clasificare. Principiul se poate enunța astfel: decizia unei populații de indivizi este mai fiabilă decât decizia unui singur individ, atât timp cât deciziile individuale ale membrilor populației sunt de o calitate decentă.

Un arbore de decizie singular se construiește pornind de la întregul set de date de antrenare, conform cu cele studiate în laboratorul anterior. Un astfel de arbore este puternic influențat de modificări ale datelor de antrenare: orice schimbare a datelor poate conduce la modificări semnificative ale structurii arborelui. De asemenea, un arbore singular este susceptibil la fenomenul de *overfitting* – arborele are eroare de clasificare redusă pe setul de date de antrenare, dar nu generalizează suficient de bine – eroare mare de clasificare pentru alte date de test.

În cazul metodei Random Forest, se folosește o multitudine de arbori de decizie, astfel:

- fiecare arbore se generează pornind de la o submulțime a datelor de antrenare inițiale și/sau folosind o submulțime a atributelor datelor (Fig. 1)
- submulțimile de date și attribute se generează aleator (instanțele și attributele se pot repeta de la o submulțime la alta)
- o instanță se clasifică separat de către fiecare arbore. Rezultă câte o decizie (câte o clasă) pentru fiecare arbore
- clasa finală este cea majoritară (clasa care apare cel mai frecvent printre mulțimea de clase furnizată de arbori)

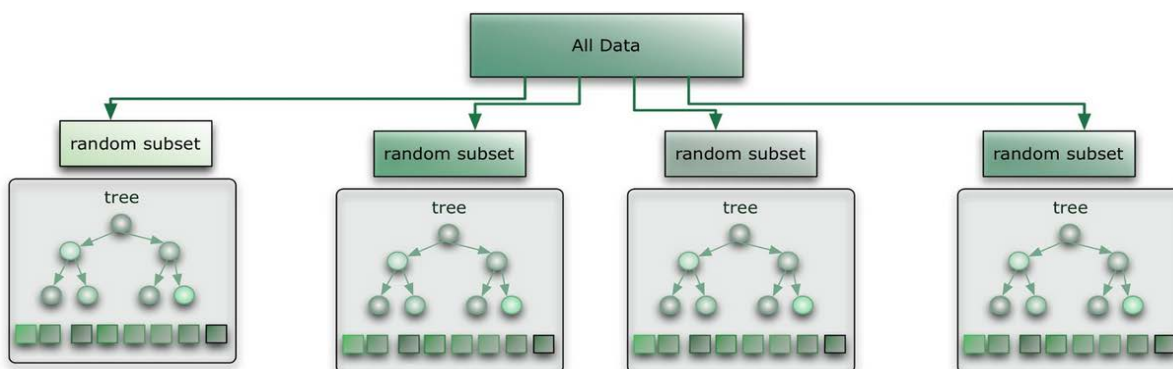


Fig.1. Generarea de arbori în cadrul algoritmului Random Forest

În cele ce urmează vom ilustra modul de generare și utilitatea unui random forest printr-un exemplu: pornim cu un set de date de antrenare (Tabelul 1) și unul de test (Tabelul 2). Vom construi un arbore singular și un random forest pornind de la aceleași date antrenare. Apoi, folosind setul de date de test, vom determina și vom compara erorile de clasificare ale arborelui singular cu cele ale random forest.

Tabelul 1. Setul de date de antrenare

Starea vremii	Temperatura	Umiditate	Vant	Joc
Ploaie	Mare	Mare	Absent	Nu
Ploaie	Mare	Mare	Prezent	Nu
Innorat	Mare	Mare	Absent	Da
Soare	Medie	Mare	Absent	Da
Soare	Mica	Normala	Absent	Da
Soare	Mica	Normala	Prezent	Nu
Innorat	Mica	Normala	Prezent	Da
Ploaie	Medie	Mare	Absent	Nu
Ploaie	Mica	Normala	Absent	Da
Soare	Medie	Normala	Absent	Da
Ploaie	Medie	Normala	Prezent	Da
Innorat	Medie	Mare	Prezent	Da
Innorat	Mare	Normala	Absent	Da
Soare	Medie	Mare	Prezent	Nu

Tabelul 2. Setul de date de test

Starea vremii	Temperatura	Umiditate	Vant	Joc
Ploaie	Mare	Mare	Absent	Da
Ploaie	Mare	Mare	Prezent	Nu
Innorat	Medie	Normala	Absent	Da
Innorat	Medie	Mare	Absent	Nu
Soare	Mica	Normala	Absent	Da
Soare	Mica	Normala	Prezent	Da
Ploaie	Mica	Normala	Prezent	Da
Ploaie	Medie	Mare	Absent	Nu
Ploaie	Mica	Normala	Absent	Da
Soare	Medie	Normala	Absent	Da
Ploaie	Mare	Normala	Prezent	Nu
Innorat	Mare	Mare	Prezent	Nu
Innorat	Mare	Normala	Absent	Da
Soare	Medie	Mare	Absent	Nu



**Arborele singular:**

Pentru obținerea arborelui, vom proceda asemănător cu generarea arborelui ID3 (laboratorul anterior), însă pentru a sorta atributele funcție de importanța lor decizională vom folosi indicele Gini, calculat conform cu Ecuația 1.

$$Gini(S) = 1 - \sum_{i=1}^C p_i^2 \quad (1)$$

Unde:

S este setul de date pentru care se calculează indicele Gini

C este numărul de clase (în setul de date din tabelul anterior sunt două clase, așadar,  $C = 2$ )

$p_i$  este probabilitatea de apariție a clasei  $i$  în setul de date S

În cazul setului de date anterior,

$$Gini(S) = 1 - (p(Da)^2 + p(Nu)^2)$$

$$p(Da) = 9/14, p(Nu) = 5/14, \text{ așadar}$$

$$Gini(S) = 1 - (9/14)^2 - (5/14)^2 = 0.459$$

Ca și în cazul arborilor ID3, pentru arborele întreg și pentru fiecare subarbore al său, trebuie să determinăm atributul care are cea mai mare importanță decizională – *atributul cu indicele Gini minim*. Calculul indicilor Gini ai atributelor se realizează pe același principiu ca și determinarea entropiei. Un atribut A cu valorile V1, V2, V3 cauzează partiționarea setului de date S în trei submulțimi SV1, SV2, SV3, fiecare corespunzând unei valori a atributului. SV1 este submulțimea lui S care conține doar instanțele cu valoarea V1 a atributului A, analog pentru SV2, SV3. În acest caz, indicele Gini care rezultă în urma partiționării datelor folosind atributul A se determină astfel:

$$Gini(S,A) = p(V1) * Gini(SV1) + p(V2) * Gini(SV2) + p(V3) * Gini(SV3)$$

Unde:

$p(V1)$  este probabilitatea ca atributul A să aibă valoarea V1

$Gini(SV1)$  este indicele Gini determinat pentru submulțimea SV1

Idem pentru V2 și V3

Ca valoare a unui nod al arborelui, se utilizează atributul cu indicele Gini minim, iar partiționarea ulterioară a datelor se realizează folosind valorile acelui atribut.

*Calcululele sunt foarte similare cu cele efectuate în cazul arborilor ID3 din laboratorul anterior, diferența esențială este utilizarea indicelui Gini în locul entropiei.*

Pentru datele din Tabelul 1 rezultă următorul arbore:

**Starea vremii**

```
--Innorat--> Da
--Soare--> Vant
                --Prezent--> Nu
                --Absent--> Da
--Ploaie--> Umiditate
                        --Normala--> Da
                        --Mare--> Nu
```

Determinăm eroarea de clasificare a acestui arbore folosind datele de test: în Tabelul 3 se observă că acest arbore nu reușește să clasifice corect anumite instanțe. Din cele 14 instanțe 6 sunt clasificate greșit, așadar **eroarea de clasificarea a arborelui este  $6/14 = 0.428$** .

Tabelul 3. Setul de date de test: coloana ST conține clasele prezise de arborele singular, iar coloana Joc conține clasele din setul de date (cele "corecte")

Starea vremii	Temperatura	Umiditate	Vant	ST	Joc
Ploaie	Mare	Mare	Absent	Nu	Da
Ploaie	Mare	Mare	Prezent	Nu	Nu
Innorat	Medie	Normala	Absent	Da	Da
Innorat	Medie	Mare	Absent	Da	Nu
Soare	Mica	Normala	Absent	Da	Da
Soare	Mica	Normala	Prezent	Nu	Da
Ploaie	Mica	Normala	Prezent	Da	Da
Ploaie	Medie	Mare	Absent	Nu	Nu
Ploaie	Mica	Normala	Absent	Da	Da
Soare	Medie	Normala	Absent	Da	Da
Ploaie	Mare	Normala	Prezent	Da	Nu
Innorat	Mare	Mare	Prezent	Da	Nu
Innorat	Mare	Normala	Absent	Da	Da
Soare	Medie	Mare	Absent	Da	Nu

### Random forest:

Presupunem că "pădurea" este formată din n arbori:

- pornind de la setul de date de antrenare (Tabelul 1), generăm aleator
  - o n submulțimi ale setului de date
  - o n submulțimi ale listei atributelor
- pentru fiecare submulțime a setului de date și submulțime de attribute vom construi un arbore separat, prin aceeași metodă care s-a utilizat la generarea arborelui singular menționat anterior

**Exemplu:** 5 arbori generați pornind de la 5 submulțimi aleatoare ale setului de date de antrenare și 5 liste ce conțin o parte din atributele datelor.

Pentru fiecare arbore Tree 1-5 se prezintă, în ordine:

- structura arborelui
- atributele care s-au luat în calcul la generarea arborelui
- submulțimea datelor de antrenare pe baza căreia s-a generat arborele

Tree 1:

```

Starea vremii
  --Soare--> Temperatura
                    --Mare--> Da
                    --Medie--> Da
                    --Mica--> Da
  --Ploaie--> Temperatura
                    --Mare--> Nu
                    --Medie--> Nu
                    --Mica--> Da
  --Innorat--> Da
    
```

['Temperatura', 'Starea vremii']

	Starea vremii	Temperatura	Umiditate	Vant	Joc
4	Soare	Mica	Normala	Absent	Da
2	Innorat	Mare	Mare	Absent	Da
1	Ploaie	Mare	Mare	Prezent	Nu
8	Ploaie	Mica	Normala	Absent	Da
12	Innorat	Mare	Normala	Absent	Da
0	Ploaie	Mare	Mare	Absent	Nu
5	Soare	Mica	Normala	Prezent	Nu
7	Ploaie	Medie	Mare	Absent	Nu
11	Innorat	Medie	Mare	Prezent	Da
9	Soare	Medie	Normala	Absent	Da
6	Innorat	Mica	Normala	Prezent	Da

Tree 2:

```

Vant
  --Absent--> Da
  --Prezent--> Temperatura
                    --Mare--> Nu
                    --Medie--> Da
                    --Mica--> Da
    
```

['Vant', 'Temperatura', 'Starea vremii']

	Starea vremii	Temperatura	Umiditate	Vant	Joc
12	Innorat	Mare	Normala	Absent	Da
2	Innorat	Mare	Mare	Absent	Da
10	Ploaie	Medie	Normala	Prezent	Da
8	Ploaie	Mica	Normala	Absent	Da
11	Innorat	Medie	Mare	Prezent	Da
1	Ploaie	Mare	Mare	Prezent	Nu

Tree 3:

```

Starea vremii
  --Soare--> Vant
                    --Absent--> Da
                    --Prezent--> Nu
  --Ploaie--> Temperatura
                    --Mare--> Da
                    --Medie--> Nu
                    --Mica--> Da
  --Innorat--> Da
    
```

## Învățare automată – Laborator 5

```
['Starea vremii', 'Vant', 'Temperatura']
```

	Starea vremii	Temperatura	Umiditate	Vant	Joc
3	Soare	Medie	Mare	Absent	Da
7	Ploaie	Medie	Mare	Absent	Nu
11	Innorat	Medie	Mare	Prezent	Da
8	Ploaie	Mica	Normala	Absent	Da
5	Soare	Mica	Normala	Prezent	Nu

Tree 4:

```
Starea vremii
  --Soare--> Vant
                --Absent--> Da
                --Prezent--> Nu
  --Ploaie--> Vant
                --Absent--> Nu
                --Prezent--> Nu
  --Innorat--> Da
```

```
['Starea vremii', 'Vant']
```

	Starea vremii	Temperatura	Umiditate	Vant	Joc
0	Ploaie	Mare	Mare	Absent	Nu
3	Soare	Medie	Mare	Absent	Da
11	Innorat	Medie	Mare	Prezent	Da
12	Innorat	Mare	Normala	Absent	Da
8	Ploaie	Mica	Normala	Absent	Da
13	Soare	Medie	Mare	Prezent	Nu
9	Soare	Medie	Normala	Absent	Da
7	Ploaie	Medie	Mare	Absent	Nu
5	Soare	Mica	Normala	Prezent	Nu
4	Soare	Mica	Normala	Absent	Da
2	Innorat	Mare	Mare	Absent	Da

Tree 5:

```
Vant
  --Absent--> Da
  --Prezent--> Temperatura
                        --Mare--> Nu
                        --Medie--> Nu
                        --Mica--> Umiditate
                                --Mare--> Da
                                --Normala--> Da
```

```
['Temperatura', 'Umiditate', 'Vant']
```

	Starea vremii	Temperatura	Umiditate	Vant	Joc
5	Soare	Mica	Normala	Prezent	Nu
2	Innorat	Mare	Mare	Absent	Da
13	Soare	Medie	Mare	Prezent	Nu
6	Innorat	Mica	Normala	Prezent	Da
4	Soare	Mica	Normala	Absent	Da
8	Ploaie	Mica	Normala	Absent	Da

Clasificăm datele din setul de antrenare folosind cei 5 arbori, individual. Clasa determinată de întreaga “pădure” va fi cea obținută de majoritatea arborilor. Comparăm apoi eroarea care rezultă în urma acestei clasificări cu cea determinată pentru arborele singular. Rezultatele sunt prezentate în Tabelul 4. Se observă că Random Forest reușește să clasifice corect mai multe instanțe decât arborele singular.

**Random Forest** a clasificat greșit 3 instanțe din cele 14, așadar **eroarea de clasificare este  $3/14 = 0.214$** . Amintim faptul **că arborele singular avea o eroare de 0.428**.

Tabelul 4. Evaluarea clasificării folosind Random Forest și un arbore singular. Coloanele T1-5 conțin clasele determinate folosind cei 5 arbori din Random Forest. Coloana RF conține decizia majoritară a celor 5 arbori. Coloana ST conține clasele determinate de arborele singular (aceleași ca în Tabelul 3). Coloana Joc conține clasele din setul de date de test (cele "corecte")

Starea vremii	Temperatura	Umiditate	Vant	T1	T2	T3	T4	T5	RF	ST	Joc
Ploaie	Mare	Mare	Absent	Nu	Da	Da	Nu	Da	Da	Nu	Da
Ploaie	Mare	Mare	Prezent	Nu	Nu	Da	Nu	Nu	Nu	Nu	Nu
Innorat	Medie	Normala	Absent	Da	Da	Da	Da	Da	Da	Da	Da
Innorat	Medie	Mare	Absent	Da	Da	Da	Da	Da	Da	Da	Nu
Soare	Mica	Normala	Absent	Da	Da	Da	Da	Da	Da	Da	Da
Soare	Mica	Normala	Prezent	Da	Da	Nu	Nu	Da	Da	Nu	Da
Ploaie	Mica	Normala	Prezent	Da	Da	Da	Nu	Da	Da	Da	Da
Ploaie	Medie	Mare	Absent	Nu	Da	Nu	Nu	Da	Nu	Nu	Nu
Ploaie	Mica	Normala	Absent	Da	Da	Da	Nu	Da	Da	Da	Da
Soare	Medie	Normala	Absent	Da	Da	Da	Da	Da	Da	Da	Da
Ploaie	Mare	Normala	Prezent	Nu	Nu	Da	Nu	Nu	Nu	Da	Nu
Innorat	Mare	Mare	Prezent	Da	Nu	Da	Da	Nu	Da	Da	Nu
Innorat	Mare	Normala	Absent	Da	Da	Da	Da	Da	Da	Da	Da
Soare	Medie	Mare	Absent	Da	Da	Da	Da	Da	Da	Da	Nu

### Cerințe:

1) Implementați metoda de generare a unui arbore pornind de la un set de date, folosind indicele Gini pentru selecția atributelor. Generați un arbore folosind setul de date data\_vreme3.csv și determinați eroarea sa de clasificare folosind datele de test din data\_vreme5.csv

2) Generați un număr oarecare n de submulțimi aleatoare ale datelor în data\_vreme3. Pentru fiecare submulțime, alegeți aleator 2 sau 3 atribute dintre cele 4 ale datelor. Generați câte un arbore pentru fiecare submulțime și listă de atribute. Determinați eroarea de clasificare a "pădurii" formate din cei n arbori, așa cum s-a demonstrat în documentația de la laborator. Comparați eroarea cu cea a arborelui determinat la 1).

## Regresie liniară și polinomială

Noțiunea de *regresie* se referă la o categorie de metode statistice care au ca scop estimarea relațiilor dintre una sau mai multe variabile independente și una sau mai multe variabile dependente. Metodele de regresie estimează valoarea unei variabile dependente asociate valorii unei variabile independente sau valorilor unui set de variabile independente. Astfel, scopul regresiei este de a determina o funcție a variabilelor independente care furnizează valoarea dependentă corespunzătoare. Tehnicile prin care se realizează estimarea în cadrul regresiei depind în principal de tipul datelor de intrare/ieșire din setul de date de antrenare (cel deja cunoscut) și tipul funcției care se estimează.

### Regresia liniară

Scopul regresiei liniare este de a determina parametrii unei funcții liniare care descrie relația dintre variabilele dependente și cele independente. Altfel spus, se dorește obținerea unei funcții de gradul I care să permită estimarea unei valori de ieșire pentru orice valoare de intrare din domeniul vizat. Estimarea se face pe baza unui set de date cu intrări/ieșiri cunoscute.

### Exemplu:

Fie următorul set de date, care conține intervalul de timp dedicat studiului pentru susținerea un examen și notele obținute de câțiva studenți la acel examen (Fig. 1):

Nr ore de studiu	0.5	0.75	1	1.25	1.5	1.75	1.75	2	2.25	2.5	2.75	3	3.25	3.5	4	4.25	4.5	4.75	5	5.5
Nota obținută	4	3	2.5	1	2	3.5	6	4	7	1.5	5	2.5	5.5	3	8	7	7.5	6	8.5	9.5

Acestea constituie datele de antrenare, scopul fiind de a determina o funcție care să permită estimarea notei obținute pentru orice număr de ore de studiu. În cazul regresiei liniare, această funcție este o dreaptă (o funcție de gradul I).

Folosim următoarele notații:

$N$  – numărul de instanțe din setul de date de antrenare (în exemplu,  $N = 20$ )

$x_i, i = 1..N$  – valorile variabilei independente (numărul de ore de studiu)

$y_i, i = 1..N$  – valorile variabilei dependente (nota)

$\bar{x}$  – valoarea medie a variabilei independente

$\bar{y}$  – valoarea medie a variabilei dependente

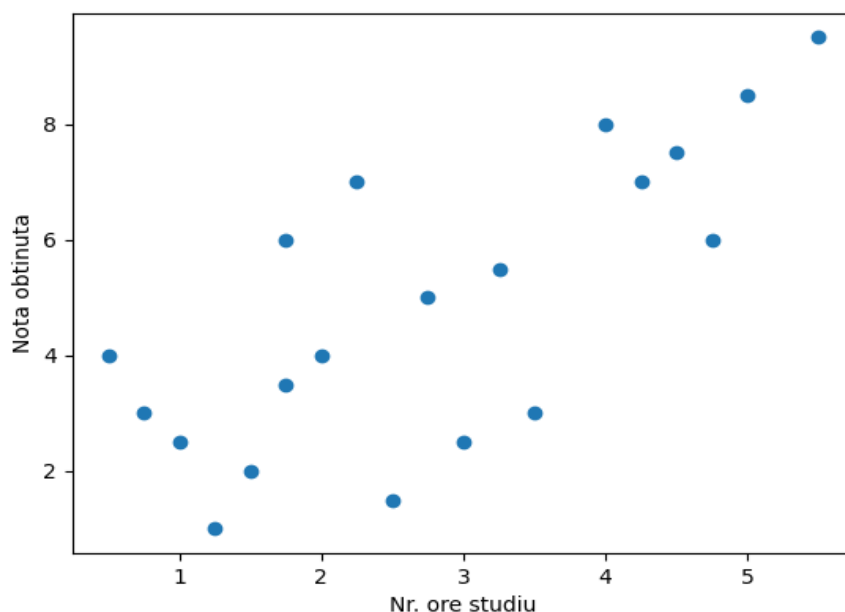


Fig. 1. Datele de antrenare

Trebuie să determinăm dreapta care exprimă cel mai bine dependența notei obținute de numărul de ore de studiu. Fie  $b_0$  și  $b_1$  coeficienții acestei drepte. Funcția pe care trebuie să o estimăm este cea din Ecuația 1. Estimarea se referă la determinarea celor mai utile valori ale coeficienților  $b_0$  și  $b_1$ . Pentru datele prezentate anterior, funcția care corespunde acestor coeficienți este reprezentată în Fig. 1

$$y = b_0 + b_1x \quad (1)$$

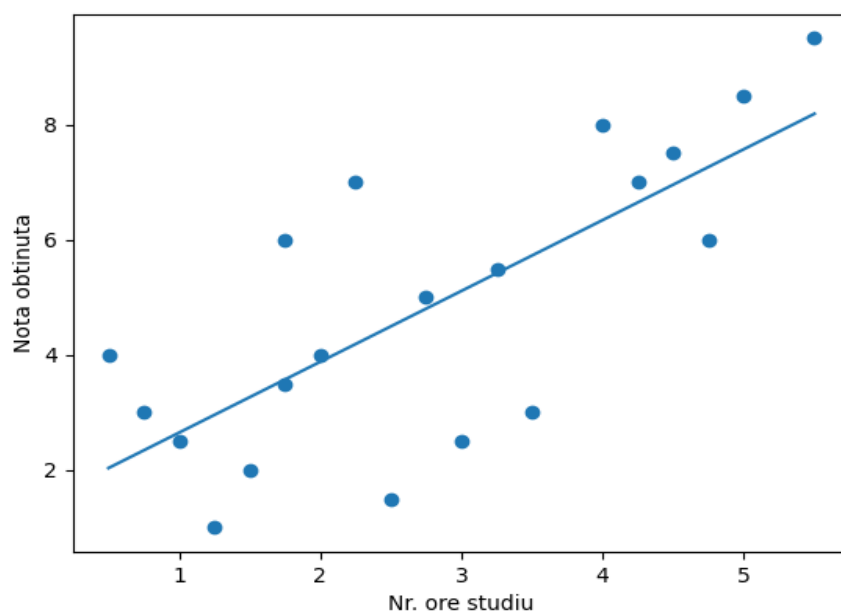


Fig. 2. Funcția liniară care descrie cel mai bine dependența dintre variabila independentă (nr. ore studiu) și cea dependentă (nota obținută)

### Soluția analitică

Fie  $\hat{y}_i$  valorile returnate de funcția căutată (estimări ale notei obținute) pornind de la valorile cunoscute ale variabilei independente (nr. orelor de studiu) (Ecuția 2).

$$\hat{y}_i = b_0 + b_1 x_i, i = 1..N \quad (2)$$

Pentru fiecare  $x_i$  obținem o eroare  $e_i = y_i - \hat{y}_i$ . Dreapta dorită trebuie să minimizeze aceste erori. Altfel spus, această dreaptă are proprietatea că diferențele dintre valorile din setul de date inițial și cele obținute cu funcția care descrie dreapta sunt minime. Astfel, funcția care trebuie minimizată este cea din Ecuția 3, care se dezvoltă în Ecuția 4:

$$S = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (3)$$

$$S = \sum_{i=1}^n (y_i - (b_0 + b_1 x_i))^2 \quad (4)$$

Așadar, scopul este de a determina coeficienții  $b_0$  și  $b_1$  care minimizează funcția din Ecuția 4. Pentru aceasta, se determină derivatele funcției  $S$  în raport cu  $b_0$  și  $b_1$  și se caută valorile coeficienților pentru care derivatele sunt nule. Efectuând calculele aferente, rezultă formulele de calcul ale celor doi coeficienți (ecuațiile 5).

$$b_0 = \bar{y} - b_1 \bar{x} \quad (5)$$

$$b_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Odată determinată funcția dorită, se poate estima “corectitudinea” acesteia determinând eroarea medie pătratică (Ecuția 6).

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (6)$$

### Soluția numerică

Soluția analitică prezentată anterior determină cu precizie valorile coeficienților. În practică, există însă numeroase situații în care coeficienții nu se pot determina prin calcule matematice directe. În aceste cazuri, se pot folosi diverse metode de optimizare numerică. Acestea presupun determinarea valorilor coeficienților pe parcursul mai multor iterații în cadrul cărora coeficienții se ajustează puțin câte puțin, până când valorile lor se apropie suficient de mult de cele ideale. Una dintre cele mai frecvent-întâlnite astfel de metode este **gradientul descendent**. Pașii sunt următorii:



- Se stabilește un număr de iterații și o rată de învățare  $\alpha$  (valoare subunitară mică)
- Se inițializează coeficienții (de exemplu cu valori aleatoare)
- Se stabilește modul de calcul al erorii. Se poate folosi *Sum of Squared Errors* (Ecuația 7)

$$SSE = \frac{1}{2} \sum_{i=1}^n (y_i - (b_0 + b_1 x_i))^2 \quad (7)$$

- Pentru fiecare iterație:
  - o Se determină eroarea (valoarea SSE).
  - o Se determină valorile componentelor gradientului (derivatele parțiale ale erorii în raport cu coeficienții). Calculul gradientului se poate face prin:
    - determinarea analitică a derivatelor parțiale (ecuațiile 8)

$$\frac{\partial SSE}{\partial b_0} = \sum_{i=1}^n -(y_i - (b_0 + b_1 x_i)) \quad (8)$$

$$\frac{\partial SSE}{\partial b_1} = \sum_{i=1}^n -(y_i - (b_0 + b_1 x_i)) x_i$$

- aproximarea derivatelor folosind diferențe finite (ecuațiile 9). ( $\Delta$  este o valoare subunitară mică)

$$\frac{\partial SSE}{\partial b_0} = \frac{SSE(b_0 + \Delta) - SSE(b_0)}{\Delta} \quad (9)$$

$$\frac{\partial SSE}{\partial b_1} = \frac{SSE(b_1 + \Delta) - SSE(b_1)}{\Delta}$$

- o Se actualizează coeficienții astfel:

$$b_0 = b_0 - \alpha \frac{\partial SSE}{\partial b_0} \quad (10)$$

$$b_1 = b_1 - \alpha \frac{\partial SSE}{\partial b_1}$$

- o Se determină noua eroare SSE. Dacă aceasta este suficient de apropiată ca valoare de eroarea de la începutul iterației, algoritmul se oprește.

### Regresia polinomială

Cazul anterior este o situație particulară, când se caută o funcție de gradul 1. În caz general, funcția dorită poate fi un polinom de orice grad:

$$y = b_0 + b_1x + b_2x^2 + b_3x^3 + \dots \quad (11)$$

Ca și în cazul regresiei liniare, coeficienții  $b_i$  se pot determina analitic sau prin metoda gradientului descendent.

### Soluția analitică

Ecuția 11 se poate scrie sub formă matriceală (Ecuția 12).

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} \quad (12)$$

$$\vec{y} = \mathbf{X}\vec{\beta}$$

Unde  $n$  este numărul instanțelor (nr valorilor din setul de date de antrenare) și  $m$  este gradul polinomului căutat. Prin prelucrarea ecuației (12) rezultă expresia vectorului de coeficienți (Ecuția 13).

$$\vec{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{y} \quad (13)$$

**Cerințe:**

## 1) Implementați regresia liniară pentru datele din exemplu

- determinați cei doi coeficienți și afișați dreapta obținută (pentru datele din exemplu, din calculul analitic ar trebui să rezulte  $b_0 = 1.419$ ,  $b_1 = 1.23$ ). Determinați eroarea medie pătratică (pentru datele din exemplu,  $MSE = 2.709$ ).
- determinați cei doi coeficienți prin metoda gradientului descendent. Calculați eroarea medie pătratică și comparați rezultatele cu cele obținute anterior (ar trebui să rezulte valori foarte apropiate de cele determinate analitic).

## 2) Implementați regresia polinomială folosind polinoame de gradul 2 ... 16.

- determinați eroarea pentru fiecare funcție polinomială.
- determinați polinomul care se potrivește cel mai bine pe datele din laborator (cel pentru care MSE este minim). În Fig 3 se prezintă câteva exemple de regresie cu polinoame de diverse grade.

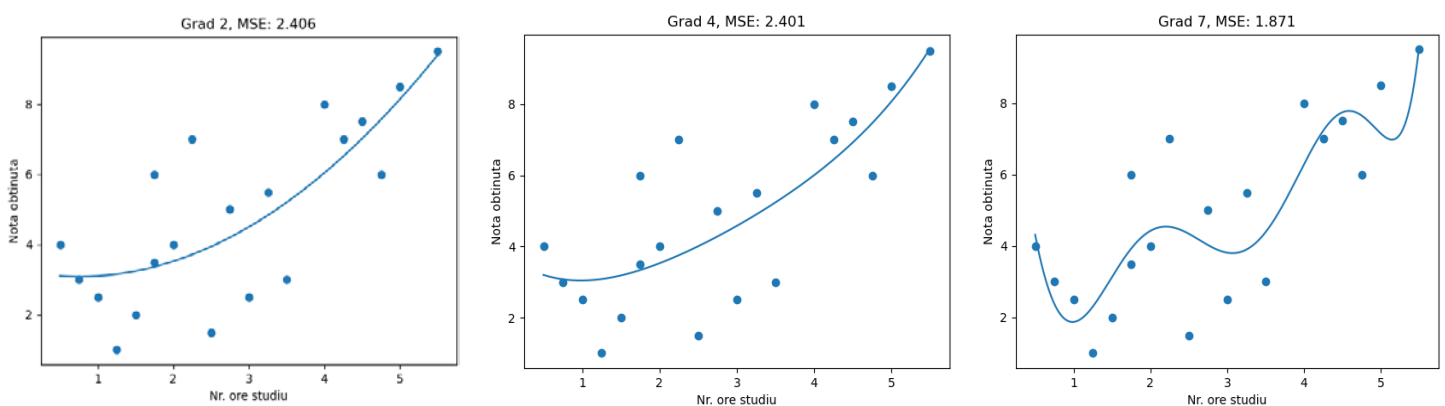


Fig. 3. Regresie folosind polinoame de diverse grade și valorile MSE care rezultă în fiecare caz

Pentru afișarea grafică a datelor și funcțiilor se poate proceda ca în exemplul următor:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1, 3, 4, 6, 3, 6, 7, 2])
y = np.array([5, 2, 5, 7, 1, 4, 3, 5])

def myFunc(x):
    return 0.7 * x ** 2 - 3 * x + 1

plt.scatter(x, y)
xplot = np.arange(min(x), max(x), 0.01)
plt.plot(xplot, myFunc(xplot))
plt.xlabel('x values')
plt.ylabel('y values')
plt.show()
```

## Regresie logistică

Noțiunea de *regresie* se referă la o categorie de metode statistice care au ca scop estimarea relațiilor dintre una sau mai multe variabile independente și una sau mai multe variabile dependente. Metodele de regresie estimează valoarea unei variabile dependente asociate valorii unei variabile independente sau valorilor unui set de variabile independente. Astfel, scopul regresiei este de a determina o funcție a variabilelor independente care furnizează valoarea dependentă corespunzătoare. Tehnicile prin care se realizează estimarea în cadrul regresiei depind în principal de tipul datelor de intrare/ieșire din setul de date de antrenare (cel deja cunoscut) și de tipul funcției care se estimează.

Metodele de regresie logistică se aplică în situațiile în care variabila dependentă este, în cel mai simplu caz, binară (Da/Nu, Adevărat/Fals, 0/1 etc). În cazul regresiei logistice, se caută o funcție care, pentru o valoare arbitrară a variabilei independente, generează o valoare dependentă în intervalul (0, 1) care exprimă probabilitatea de încadrare în cele două cazuri descrise de valorile inițiale ale variabilei dependente.

### Exemplu:

Următorul set de date conține numărul de ore dedicat studiului și valorile de adevăr ale afirmației “Am promovat examenul.”

Nr ore de studiu	0.5	0.75	1	1.25	1.5	1.75	1.75	2	2.25	2.5	2.75	3	3.25	3.5	4	4.25	4.5	4.75	5	5.5
Promovare	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

Dorim să determinăm o funcție care să permită determinarea probabilității de promovare a examenului pentru orice valoare a numărului de ore de studiu. Această funcție are expresia din Ecuația 1. Pentru anumite valori ale parametrilor  $w_0$  și  $w_1$ , se obține graficul din Fig. 1, unde se reprezintă și datele de antrenare.

$$y = \frac{1}{1 + e^{-(w_0 + w_1 x)}} \quad (1)$$

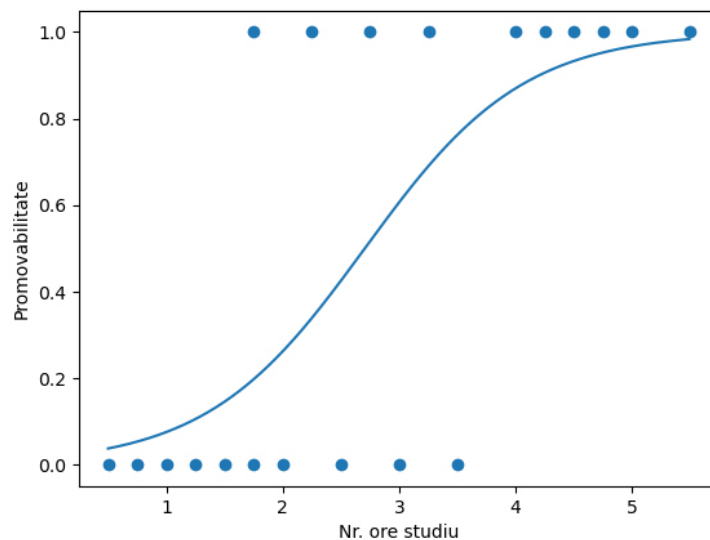
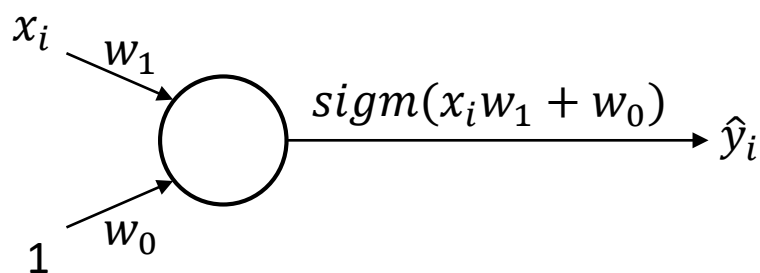


Fig. 1. Funcția care rezultă prin aplicarea regresiei logistice pe datele de antrenare, pentru  $w_0 = -3.966$ ,  $w_1 = 1.467$

Scopul este de a determina valorile coeficienților  $w_0$  și  $w_1$ , astfel încât funcția din Ecuația 1 să genereze erori cât mai mici pentru datele de antrenare. Dorim o pereche de valori ale coeficienților care să minimizeze diferențele dintre probabilitățile obținute cu funcția din Ecuația 1 și valorile de adevăr din setul de date, pentru toate valorile variabilei independente (ale numărului de ore de studiu). Spre deosebire de cazul regresiei liniare, în situația dată nu există o formulă analitică de calcul a acestor coeficienți, valorile lor trebuind estimate. Dintre metodele de determinare a coeficienților, o propunem pe următoarea:

Definim un perceptron de forma:



Unde  $x_i$  sunt valorile variabilei independente (orele de studiu),  $w_0$  și  $w_1$  sunt ponderile intrărilor perceptronului, iar funcția de activare este funcția sigmoid (Ecuația 2).

$$\text{sigm}(x) = \frac{1}{1+e^{-x}} \quad (2)$$

**Antrenarea perceptronului se realizează astfel:**

Pentru determinarea erorii, vom folosi funcția *cross entropy* (Ecuația 3). Pe parcursul mai multor iterații, vom determina valoarea acestei funcții (eroarea de la fiecare iterație), vom determina gradientii erorii în raport cu cele două ponderi  $w_1$  și  $w_0$  (Ecuația 4) și vom actualiza ponderile conform cu Ecuația 5, în ideea de a reduce eroarea la fiecare iterație.

$$CE = -\frac{1}{N} \sum_{i=0}^{N-1} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \quad (3)$$

$$\frac{\partial CE}{\partial w_0} = \hat{y} - y \quad (4)$$

$$\frac{\partial CE}{\partial w_1} = (\hat{y} - y)x$$

...unde

$x$  sunt valorile variabilei independente (nr. ore studiu)

$y$  sunt valorile variabilei dependente (valorile de 0 sau 1 ale promovării)

$\hat{y}$  sunt valorile predicțiilor care se obțin pentru  $x$  (valorile care se obțin cu funcția (1) )

- se inițializează ponderile  $w_0$  și  $w_1$  cu valori aleatoare dintr-un interval restrâns, de ex  $[-0.5, 0.5]$
- se stabilește un număr de iterații (o valoare mare, de ex. 1000), în cadrul fiecărei iterații având loc o etapă de calcul a ieșirii  $\hat{y}$  și de reglare a ponderilor (etapă numită și “epocă”)
- pentru fiecare astfel de iterație
  - o se determină eroarea (Ecuația 3)
  - o se determină valoarea de la ieșirea perceptronului
  - o se determină componentele gradientului (Ecuația 4)
  - o se actualizează ponderile (Ecuația 5). Parametrul  $\alpha$  se numește *rată de învățare* și are de obicei o valoare subunitară mică (se poate încerca inițial  $\alpha = 0.01$  și apoi, dacă este cazul, se ajustează acest parametru în ideea de a obține o eroare cât mai mică)
  - o se determină din nou eroarea și se verifică dacă aceasta se apropie suficient de mult de valoarea determinată la început (i.e. dacă modulul diferenței erorilor este mai mic decât o constantă subunitară  $\epsilon$  de valoare mică (de ex. 0.001)). Dacă erorile sunt suficient de apropiate ca valoare, se consideră algoritmul încheiat și se consideră corecte valorile curente ale  $w_0$  și  $w_1$ .

$$w_0 = w_0 - \alpha \frac{\partial CE}{\partial w_0} \quad (5)$$

$$w_1 = w_1 - \alpha \frac{\partial CE}{\partial w_1}$$

### Aplicații:

1. Implementați regresia logistică pentru datele din exemplu:

- Determinați cei doi coeficienți și funcția obținută (pentru verificare, un set de valori utile este  $w_0 = -3.966$ ,  $w_1 = 1.467$ )
  - o Pentru afișarea graficelor în Python, vezi laboratorul anterior
- Determinați probabilitatea de promovare a examenului pentru o valoare oarecare a numărului de ore de studiu
- Afișați un grafic care să ilustreze evoluția erorii pe parcursul iterațiilor.

2. Implementați regresia logistică folosind ca eroare funcția MSE (Ecuația 6). Determinați gradientii aproximându-i prin diferențe finite (vezi laboratorul anterior). Ajustați parametrii din faza de antrenare astfel încât să obțineți valori cât mai apropiate de cele de la 1).

## Algoritmi de boosting. AdaBoost

Algoritmii de *boosting* constau în metode bazate pe ansambluri ce au ca scop crearea unui clasificator puternic (*strong classifier*) folosind rezultatele mai multor clasificatori slabi (*weak classifier*). Un clasificator slab generează rezultate cu acuratețe redusă (eroarea de clasificare este semnificativă, peste 20–30%), dar, prin combinarea adecvată a mai multor astfel de clasificatori, eroarea de clasificare se poate reduce semnificativ.

AdaBoost este unul din cei mai populari algoritmi de boosting. Metoda începe cu o etapă de antrenare în cadrul căreia un clasificator slab se aplică în repetate rânduri pe un set de date de antrenare (o mulțime de instanțe cărora li se cunoaște deja clasa de încadrare). Fiecărei instanțe  $i$  se asociază câte o pondere, inițial aceste ponderi fiind egale. În cadrul unei iterații a algoritmului AdaBoost se determină eroarea clasificatorului slab și se ajustează ponderile astfel încât instanțele clasificate greșit să aibă ponderi mai mari. Astfel, la următoarea iterație, clasificatorul slab va prioritiza instanțele clasificate incorect, tendința fiind de a reduce eroarea de clasificare. Rezultatul final al algoritmului este o combinație liniară a rezultatelor individuale ale clasificatorilor slabi de la fiecare iterație, ponderate de erorile fiecărui clasificator slab.

### Decision Stump

Clasificatorul slab pe care îl vom utiliza și îmbunătăți este un arbore cu un singur nivel numit Decision Stump (prescurtat DS). În cadrul unui DS se ia o singură decizie funcție de un singur atribut al setului de date vizat. Astfel, un DS este un arbore de decizie care are doar rădăcina și un singur set de noduri frunză. Decizia singulară care se adoptă în cazul unui DS este adesea insuficientă pentru a realiza o clasificare riguroasă, așadar eroarea de clasificare a unui DS este în general, semnificativă.

În continuare, vom utiliza DS binari, deoarece AdaBoost este gândit, la rândul său, în ideea de a oferi decizii binare. Un astfel de arbore binar necesită un atribut funcție de care se ia decizia și o valoare  $a$  sa care împarte spațiul de instanțe în două părți a căror dimensiune depinde de valoarea aleasă.

Considerăm setul de date din Tabelul 1. Pe baza acestuia, vom genera un DS cu două ramuri, care va subdiviza setul de date funcție de valoarea medie a unuia dintre atributele  $A_1$ ,  $A_2$ . Pentru a determina atributul utilizat, vom calcula indicele Gini al fiecărui atribut (Ec. 1) și îl vom alege pe cel cu indicele minim.

$$Gini(S) = 1 - \sum_{i=1}^C p_i^2 \quad (1)$$

Unde:

$S$  este setul de date pentru care se calculează indicele Gini



C este numărul de clase (în setul de date din tabelul anterior sunt două clase, așadar,  $C = 2$ )  
 $p_i$  este probabilitatea de apariție a clasei  $i$  în setul de date S)

Tabelul 1. Set de date cu două atribute numerice. Instanțele se încadrează în două clase cu etichetele -1 și 1

A1	A2	Class
4.2	4	-1
2.1	2	-1
6	5	1
6	5.5	-1
1.5	6.5	1
3.7	7.5	1
1.5	3.2	1
4	3.5	-1
1.7	7.8	1
3.5	5.5	-1

Fie  $\bar{A}_1$  și  $\bar{A}_2$  valorile medii ale celor două atribute. Presupunem că A este atributul din nodul DS. Astfel, DS împarte setul de date în două partiții, care corespund  $A < \bar{A}$ , respectiv  $A \geq \bar{A}$ . Dorim să determinăm  $A \in \{ \}$  pentru care dintre atributele A1, A2 partiționarea generează submulțimi care minimizează indicele Gini.

Pentru A1:

$$\text{GiniA1}_{A1 < \bar{A}_1} = 1 - P^2(\text{Class} = -1 \mid A1 < \bar{A}_1) - P^2(\text{Class} = 1 \mid A1 < \bar{A}_1)$$

$$\text{GiniA1}_{A1 \geq \bar{A}_1} = 1 - P^2(\text{Class} = -1 \mid A1 \geq \bar{A}_1) - P^2(\text{Class} = 1 \mid A1 \geq \bar{A}_1)$$

$$\text{GiniA1} = P(A1 < \bar{A}_1) * \text{GiniA1}_{A1 < \bar{A}_1} + P(A1 \geq \bar{A}_1) * \text{GiniA1}_{A1 \geq \bar{A}_1}$$

Din calcule rezultă  $\text{GiniA1} = 0.416$ . În mod similar,  $\text{GiniA2} = 0.48$ . Prin urmare, DS va fi arborele reprezentat în Fig.1.

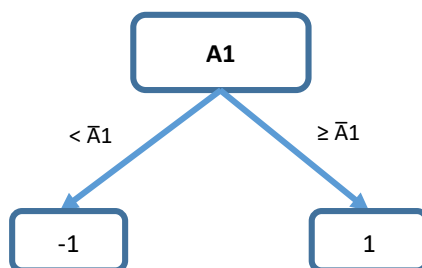


Fig 1. Decision Stump pentru datele din Tabelul 1

Eroarea de clasificare se determină aplicând DS pe datele de antrenare. Rezultatul este ilustrat în Tabelul 2, unde se observă faptul că eroarea de clasificare este 0.7.

Tabelul 2. Rezultatul clasificării folosind decision stump-ul din Fig. 1

A1	A2	Class	DS
4.2	4	-1	1
2.1	2	-1	-1
6	5	1	1
6	5.5	-1	1
1.5	6.5	1	-1
3.7	7.5	1	1
1.5	3.2	1	-1
4	3.5	-1	1
1.7	7.8	1	-1
3.5	5.5	-1	1

### AdaBoost

Pentru îmbunătățirea rezultatului anterior, vom aplica DS în repetate rânduri pe setul de date de antrenare, în cadrul mai multor iterații. Vom asigna câte o pondere pentru fiecare instanță din setul de date. Inițial, toate instanțele vor avea aceeași pondere,  $1/n$ , unde  $n = \text{nr. de instanțe}$  (Tabelul 3).

Tabelul 3. Setul de date ponderat. Fiecărei instanțe  $i$  se asociază o pondere inițializată cu  $1/\text{nr\_instanțe}$

A1	A2	W	Class
4.2	4	0.1	-1
2.1	2	0.1	-1
6	5	0.1	1
6	5.5	0.1	-1
1.5	6.5	0.1	1
3.7	7.5	0.1	1
1.5	3.2	0.1	1
4	3.5	0.1	-1
1.7	7.8	0.1	1
3.5	5.5	0.1	-1

La fiecare iterație, ponderile se vor modifica astfel încât instanțele clasificate greșit la iterația  $t-1$  să aibă ponderi mai mari la iterația  $t$ . Astfel, instanțele care au generat erori la iterația  $t-1$  vor fi luate în considerare în măsură mai mare la iterația  $t$ , în ideea de a reduce eroarea de clasificare.

Considerăm următoarele notații:

$T$  – numărul de iterații

$t$  – o iterație oarecare,  $t = 1..T$

$x$  – o instanță oarecare (un rând din tabelele cu setul de date)

$X$  – mulțimea instanțelor din setul de date

$n$  – numărul de instanțe din setul de date de antrenare

$x_i$  – o instanță din setul de date,  $x_i \in X$ ,  $i = 1..n$

$y$  – valoarea asociată unei clase:  $y \in \{-1, 1\}$

$y_i$  – clasa instanței  $x_i$  din setul de date de antrenare

$w_t(i)$  – ponderea de la iterația  $t$  a instanței  $x_i$

$h_t(x)$  – rezultatul clasificării instanței  $x$  folosind clasificatorul slab (altfel spus, clasa care se obține aplicând DS pe  $x$ ), care poate avea valorile 1 sau -1.

Algoritmul AdaBoost poate fi descris astfel:

Datele inițiale sunt de forma  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , unde  $x_i \in X$ , și  $y_i \in \{-1, 1\}$ ,  $i = 1..n$

Se inițializează ponderile  $w_1(i) = 1/n$

Se generează un prim DS conform celor descrise anterior.

Pentru  $t = 1..T$ :

- Se aplică DS pe datele de antrenare  $x_i$  și se obțin valorile pentru  $h_t(x_i)$
- Se determină eroarea ponderată a clasificatorului (suma ponderilor instanțelor clasificate greșit / suma tuturor ponderilor)

$$\varepsilon_t = P_{i \sim w_t}(h_t(x_i) \neq y_i)$$

- Se determină un coeficient  $\alpha_t$  al iterației curente astfel:

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

- Se actualizează ponderile astfel:

$$w_{t+1}(i) = w_t(i) e^{-\alpha_t y_i h_t(x_i)}$$

- Se normalizează ponderile astfel obținute (se împarte fiecare pondere la suma tuturor ponderilor)
- Se generează un nou DS cu noile valori ale ponderilor

Pentru o instanță  $x$  oarecare, clasa se determină ca fiind o combinație a tuturor DS-urilor generate în decursul iterațiilor, astfel:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

Rezultatele aplicării AdaBoost pe datele din laborator pentru trei iterații (suficiente pentru a minimiza eroarea) se prezintă în Tabelul 4.

Tabelul 4. Rezultatul aplicării AdaBoost în decursul a trei iterații.  $W_0$  sunt ponderile inițiale,  $AB_0$  sunt predicțiile generate inițial (aceleași ca în Tabelul 2).  $W_i$  și  $AB_i$ ,  $i=1..3$ , sunt ponderile, respectiv predicțiile aferente celor trei iterații

A1	A2	$W_0$	$W_1$	$W_2$	$W_3$	Class	$AB_0$	$AB_1$	$AB_2$	$AB_3$
4.2	4	0.1	0.084	0.065	0.044	-1	1	1	-1	-1
2.1	2	0.1	0.137	0.107	0.072	-1	-1	-1	-1	-1
6	5	0.1	0.137	0.185	0.125	1	1	1	-1	1
6	5.5	0.1	0.084	0.113	0.185	-1	1	1	1	1
1.5	6.5	0.1	0.084	0.065	0.044	1	-1	-1	1	1
3.7	7.5	0.1	0.137	0.107	0.072	1	1	1	1	1
1.5	3.2	0.1	0.084	0.113	0.185	1	-1	-1	-1	-1
4	3.5	0.1	0.084	0.065	0.044	-1	1	1	-1	-1
1.7	7.8	0.1	0.084	0.065	0.044	1	-1	-1	1	1
3.5	5.5	0.1	0.084	0.113	0.185	-1	1	1	1	1
Eroare							0.7	0.7	0.4	0.3

### Cerințe

1. Implementați un arbore DS. Aplicați-l pe setul de date din laborator și determinați eroarea de clasificare.
2. Implementați AdaBoost folosind DS pe post de clasificator slab. Afișați erorile clasificatorului slab de la fiecare iterație, precum și eroarea finală de clasificare a algoritmului.

**Observație:** Atunci când instanțele din setul de date sunt ponderate, toate calculele efectuate se vor realiza ponderat. Aceasta implică următoarele:

Probabilitățile se calculează astfel:

- neponderat:  $P = \frac{\text{numar\_cazuri\_favorabile}}{\text{numar\_cazuri\_posibile}}$

- ponderat:  $P = \frac{\text{suma\_ponderilor\_cazurilor\_favorabile}}{\text{suma\_ponderilor\_cazurilor\_posibile}}$

Media unui set de valori  $x_i$ , cu ponderile  $w_i$ ,  $i \in \{1, \dots, n\}$ :

- neponderat:  $\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$

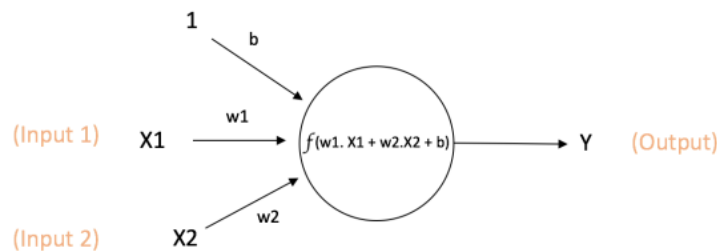
- ponderat:  $\bar{x} = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}$

## Clasificare folosind rețele neuronale simple

În cadrul unei metode de clasificare se urmărește încadrarea unor date cu anumiți parametri într-o categorie, numită și clasă, dintr-o mulțime finită și cunoscută de clase. Spre exemplu, pentru rezolvarea unei probleme de recunoaștere facială se dorește stabilirea corespondenței dintre mulțimea trăsăturilor faciale care pot fi extrase dintr-o imagine și o mulțime finită de persoane cunoscute. Algoritmii de clasificare realizează asocieri între spațiul parametrilor datelor ce se doresc clasificate și mulțimea claselor vizate. Cei mai eficienți algoritmi de clasificare sunt supervizați, ceea ce presupune o etapă de antrenare folosind date deja disponibile, algoritmi realizând asocieri între parametrii datelor de antrenare și clasele în care au fost încadrate acestea.

O rețea neuronală artificială (ANN) este un model computațional inspirat din modul în care rețelele neuronale biologice procesează informațiile în creierului uman. Rețelele neuronale artificiale se utilizează la scară largă pentru o multitudine de aplicații de cercetare și dezvoltare din domeniul învățării automate, datorită eficacității dovedite de acestea pentru rezolvarea unor probleme de recunoaștere a vorbirii, vedere artificială și procesarea textului.

Unitatea de bază a unei rețele neuronale este neuronul, care poate primi date de intrare de alți neuroni sau de la o sursă externă, pe baza căroră calculează o valoare la ieșire. Fiecare intrare are o pondere asociată ( $w$ ), care este atribuită pe baza importanței sale relativ la alte intrări. Neuronul aplică o funcție de activare  $f$  pe suma ponderată a intrărilor sale:



$$\text{Output of neuron} = Y = f(w1.X1 + w2.X2 + b)$$

Rețeaua de mai sus are intrările  $X1$  și  $X2$  și ponderile  $w1$  și  $w2$  asociate acestor intrări. În plus, există o altă intrare  $1$  cu ponderea  $b$  (numită *bias*). Ieșirea  $Y$  a neuronului se calculează ca fiind suma intrărilor  $x_i$ , ponderate de ponderile  $w_i$ . Funcția  $f$  se numește funcție de activare și trebuie să fie neliniară. Așadar unul dintre scopurile funcției de activare este introducerea neliniarității în calculul valorii de ieșire a neuronului. Acest lucru este important deoarece majoritatea datelor din lumea reală sunt neliniare, prin urmare dorim ca neuronii să învețe

reprezentări neliniare a acestora. Fiecare funcție de activare (sau non-liniaritate) primește o singură valoare și execută o anumită operație matematică. În majoritatea cazurilor se folosesc următoarele funcții de activare:

**Sigmoid:** preia o valoare de intrare reală și o aduce în intervalul (0,1)

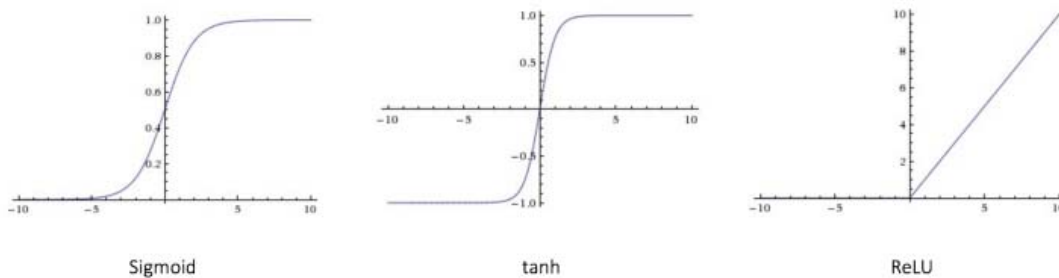
$$\sigma(x) = 1 / (1 + \exp(-x))$$

**tanh:** preia o valoare de intrare reală și o aduce în intervalul [-1, 1]

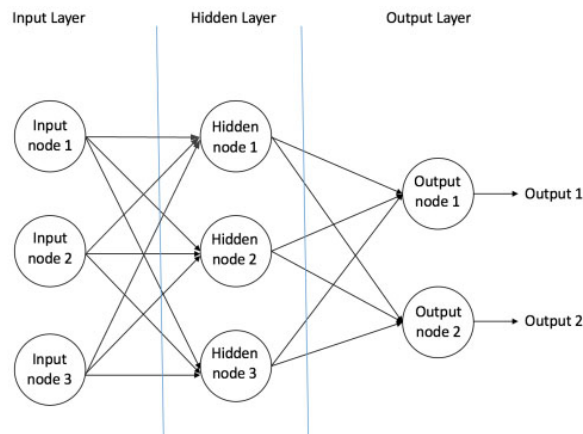
$$\tanh(x) = 2\sigma(2x) - 1$$

**ReLU:** Rectifier Linear Unit. Aplică o valoare prag 0 pe valoarea reală de intrare (înlocuiește valorile negative cu zero).

$$f(x) = \max(0, x)$$



Rețelele neuronale clasice sunt constituite din două sau mai multe straturi, fiecare cu cel puțin un neuron. Aceste rețele neuronale se mai numesc și complet conectate (fully-connected), deoarece există câte o conexiune de la fiecare neuron dintr-un strat către fiecare neuron din următorul strat. La nivelul fiecărui neuron au loc operațiile descrise anterior. Având în vedere direcția de efectuare a calculelor în cadrul unei astfel de rețele, ea mai poartă denumirea de rețea cu propagare înainte (feed forward). Într-o astfel de rețea datele de la intrare sunt supuse calculelor aferente începând cu primul strat, continuând cu straturile următoare în ordine și încheind cu determinarea valorilor stratului de ieșire.



Pentru rezolvarea problemelor de clasificare, rețelele neuronale sunt supuse unui proces de antrenare, în cadrul căruia ele învață corelații între datele de antrenare și clasele în care sunt încadrate acestea. Propagarea înapoi a erorilor (backpropagation, retropropagare) este una dintre metodele cel mai frecvent utilizate pentru antrenarea unei rețele neuronale. Este o metodă de învățare supervizată, ceea ce înseamnă învățarea se realizează pe baza unor date de antrenare deja clasificate (fiecare instanță a datelor de antrenare este etichetată ca aparținând unei anumite clase). Algoritmul backpropagation se bazează pe învățarea din greșeli, el efectuând corecții asupra rețelei prin ajustarea ponderilor acestora. Așadar scopul învățării este determinarea acelor valori ale ponderilor rețelei care să asigure realizarea unei corelații cât mai bune între datele de antrenare și etichetele acestora.

Inițial, toate ponderile iau valori aleatoare. Fiecare instanță din setul de date de antrenare se propagă înainte prin rețea, rezultând un set de valori la ieșirea rețelei. Această ieșire este comparată cu ieșirea dorită, deja cunoscută (din eticheta datelor de antrenare corespunzătoare), iar eroarea este „propagată” înapoi la nivelul anterior. Ponderile rețelei se ajustează în sensul minimizării acestei erori. Procesul se repetă până când eroarea de ieșire se situează sub un prag prestabilit. Odată ce algoritmul se încheie, rețeaua neuronală este capabilă să proceseze date de intrare noi, pe care nu le-a mai întâlnit. Corectitudinea încadrării acestor date în clasa corespunzătoare depinde de numeroși factori, cum ar fi relevanța datelor folosite la antrenare, algoritmul de optimizare folosit, rata de învățare etc.

**Aplicație:**

Dorim să proiectăm un algoritm de clasificare care să permită prelucrarea datelor din domeniul botanicii. Astfel, vom implementa o rețea neuronală cu ajutorul căreia vom determina speciile unor plante pe baza unor caracteristici biologice. Vom împărți plantele pe trei specii cunoscute, așadar urmărim asignarea plantelor în trei clase. Avem nevoie de o serie de date de antrenare (caracteristici ale plantelor și specia corespunzătoare fiecărui set de caracteristici), pe care le regăsim în fișierul **iris.csv**:

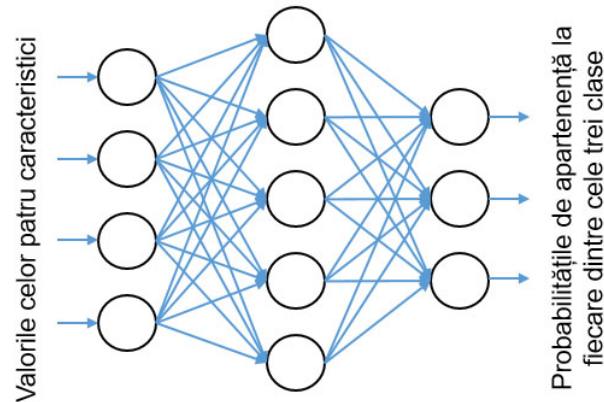
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
...				
7	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4	1.3	versicolor
...				
6.3	3.3	6	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3	5.9	2.1	virginica
...				
6.3	2.9	5.6	1.8	virginica

Plantele vizate au 4 caracteristici specificate prin valori reale, și ele se pot încadra în trei specii (trei clase). Datele de antrenare sunt în număr de 150, pentru fiecare cunoscându-se valorile caracteristicilor, precum și clasa (specia) în care se încadrează. Scopul este de a construi o rețea neuronală și de a o antrena folosind datele cunoscute, în ideea de a putea identifica specia oricărei plante cu alte valori ale acelorași caracteristici.

Fiecărei clase  $i$  se asociază câte un label (valoare numerică =  $[0 \dots nrClase-1]$ ). Așadar, cele trei clase vor avea label-urile *setosa* = 0, *versicolor* = 1, *virginica* = 2. Rețeaua neuronală va opera cu aceste label-uri în locul string-urilor aferente fiecărei clase.



Rețeaua pe care o vom implementa are următoarea structură:



Codul care implementează, antrenează și evaluează o astfel de rețea este ilustrat mai jos. Codul sursa este redactat în limbajul Python și se folosește biblioteca PyTorch. Prima dată listăm codul care implementează rețeaua neuronală, apoi îl vom parcurge pas cu pas:

```
class SimpleNN(nn.Module):
    def __init__(self, inputSize, hiddenSize, outputSize):
        super(SimpleNN, self).__init__()

        self.fc1 = torch.nn.Linear(inputSize, hiddenSize)
        self.fc2 = torch.nn.Linear(hiddenSize, outputSize)

    def forward(self, input):
        output = self.fc1(input)
        output = F.relu(output)
        output = self.fc2(output)

        return output

    def train(self, inputs, targets):
        lossFunc = nn.CrossEntropyLoss()
        nrEpochs = 10
        learnRate = 0.01
        optimizer = torch.optim.SGD(self.parameters(), learnRate)

        for epoch in range(nrEpochs):
            accuracy = 0

            for input, target in zip(inputs, targets):
                optimizer.zero_grad()
                predicted = self.forward(input.unsqueeze(0))
                loss = lossFunc(predicted, target.unsqueeze(0))
                loss.backward()
                optimizer.step()

            print('Epoch', epoch, 'loss', loss.item())
```

**Semnificația liniilor de cod este următoarea:**

```
class SimpleNN(nn.Module):
```

Rețeaua neuronală este organizată sub forma unui modul. Un modul în Pytorch implementează o serie de operații predefinite și poate la rândul său conține mai multe submodule.

```
def __init__(self, inputSize, hiddenSize, outputSize):
```

Constructorul primește ca parametri dimensiunile straturilor rețelei.

```
self.fc1 = torch.nn.Linear(inputSize, hiddenSize)
self.fc2 = torch.nn.Linear(hiddenSize, outputSize)
```

Clasa definită anterior constituie un modul care include două submodule, fc1 și fc2 (fc = "fully connected"). Acestea implementează operațiile liniare care au loc între două straturi succesive ale rețelei neuronale (suma ponderată a intrărilor). Ponderile rețelei sunt parametri predefiniți ai acestor două module.

În cazul modului fc1, se pornește cu datele din stratul de intrare (inputSize neuroni) și se realizează suma ponderată a acestora, rezultând hiddenSize valori în stratul ascuns al rețelei. Modulul fc2 preia valorile din stratul ascuns și realizează operațiile liniare aferente stratului de ieșire, rezultând outputSize valori.

```
def forward(self, input):
    output = self.fc1(input)
    output = F.relu(output)
    output = self.fc2(output)
    return output
```

Metoda implementează propagarea înainte prin rețeaua neuronală. Pytorch prelucrează datele numerice folosind tensori. Acești tensori pot fi valori scalare, vectori sau matrici bi- și multi-dimensionale. Putem considera că tensorii sunt modalitatea Pytorch de gestiune a array-urilor de valori numerice.

Parametrul **input** este un tensor ce conține valorile de la intrarea rețelei neuronale (de exemplu, un set de patru caractéristici ale unei specii de plante). **input** este supus prelucrărilor care au loc la nivelul rețelei. Prima dată se realizează operațiile din primul strat al rețelei. Pentru neuronii din stratul ascuns, se determină sumele ponderate ale valorilor din stratul de intrare (fc1(input)), cărora li se aplică funcția de activare *relu*, descrisă anterior. Apoi, valorile ce rezultă în stratul ascuns sunt supuse procesărilor liniare de unde rezultă valorile de ieșire. Astfel, la ieșirea rețelei rezultă un tensor ce conține trei valori ce reprezintă scoruri de apartenență a datelor de intrare (cele 4 caracteristici ale plantei) la cele trei clase vizate (cele

trei specii de plante). Prin normalizarea acestor scoruri se pot obține probabilitățile de apartenență la cele trei clase, dar în exemplul furnizat nu este nevoie de această normalizare.

```
def train(self, inputs, targets):
```

Această metodă realizează antrenarea rețelei. Algoritmul de antrenare folosește date de antrenare (**inputs**) pentru care se cunosc clasele în care se încadrează (**targets**).

**inputs** conține valorile caracteristicilor plantelor (așadar în exemplul furnizat este un tensor de dimensiune [150, 4] cu valori reale, 150 instanțe x 4 caracteristici)

**targets** conține câte un label pentru fiecare instanță – pentru fiecare set de 4 caracteristici din **inputs**, **targets** conține câte o valoare întreagă corespunzătoare uneia din cele trei clase disponibile. În exemplul furnizat, **targets** este un tensor de dimensiune [150, 1] ce conține valori întregi din mulțimea {0, 1, 2}, aceste valori corespunzând celor trei clase disponibile.

```
lossFunc = nn.CrossEntropyLoss()
```

Funcția cu care se evaluează eroarea dintre valorile furnizate de rețeaua neuronală și labelurile din setul de date de antrenare.

```
nrEpochs = 10
```

Numărul de epoci – o epocă este o etapă din faza de antrenare a rețelei în cadrul căreia întregul set de date de antrenare a fost folosit pentru ajustarea parametrilor rețelei. Așadar, în faza de antrenare rețeaua “vede” datele de antrenare de **nrEpochs** ori.

```
learnRate = 0.01
```

Rata de învățare = parametru care controlează măsura în care se modifică ponderile rețelei în faza de antrenare. Cu cât **learnRate** este mai mare, cu atât parametrii rețelei variază mai multe de la o fază de antrenare la alta.

```
optimizer = torch.optim.SGD(self.parameters(), learnRate)
```

Metoda de optimizare este algoritmul de ajustare al ponderilor rețelei în urma apariției unei erori între valoarea furnizată de rețea și cea din setul de date. În cazul nostru, metoda folosită se numește Stochastic Gradient Descent (SGD).

```
for epoch in range(nrEpochs):
```

Antrenarea se realizează pe parcursul mai multor epoci.

```
for input, target in zip(inputs, targets):
```

La fiecare epocă, întregul set de date de antrenare va fi supus procesărilor din faza de antrenare a rețelei.

```
optimizer.zero_grad()
```

Optimizarea presupune căutarea valorilor ponderilor rețelei care minimizează eroarea generată de rețea. Pentru aceasta, metoda de căutare se folosește de variațiile erorii în raport cu ponderile rețelei, exprimate prin componentele gradientului erorii în raport cu ponderile. Aceste componente ale gradientului trebuie resetate la începutul procesului de re-ajustare a ponderilor.

```
predicted = self.forward(input.unsqueeze(0))
```

**predicted** este rezultatul generat de rețeaua neuronală pentru valoarea dată ca parametru de intrare. În cazul de față **predicted** este un tensor ce conține trei valori, câte una pentru fiecare clasă. Cu cât valoarea aferentă unei clase este mai mare, cu atât se consideră ca rețeaua asociază acea clasă cu datele de intrare într-o măsură mai mare. **Input** este un tensor de dimensiune [4], dar, întrucât funcția de calcul a erorii utilizează tensori bidimensionali, se folosește metoda **unsqueeze** pentru a adăuga o dimensiune suplimentară tensorului (dintr-un tensor 1D de dimensiune [4] devine un tensor 2D de dimensiune [1, 4] iar valorile nu se modifică).

```
loss = lossFunc(predicted, target.unsqueeze(0))
```

Se determină diferența dintre valoarea generată de rețea și label-ul din setul de date, folosind funcția definită anterior.

```
loss.backward()
```

```
optimizer.step()
```

Se realizează propagarea înapoi a erorii prin rețeaua neuronală și se ajustează ponderile rețelei în sensul minimizării erorii, cu metoda implementată în optimizer.

### Cerințe:

1. Determinați și afișați acuratețea rețelei la sfârșitul fiecărei epoci de antrenare

Acuratețea rețelei pentru epoca curentă se determină ca fiind probabilitatea ca datele de antrenare să fi fost clasificate corect în cadrul acelei epoci (numărul de instanțe din setul de date clasificate corect / numărul total al instanțelor).

Explicații:

Pentru un input de dimensiune 4, reprezentând valorile celor 4 caracteristici ale plantelor, rețeaua generează un output de dimensiune 3, plantele încadrându-se într-una cele trei clase disponibile.

De exemplu, presupunem că pentru setul de caracteristici [7, 3.2, 4.7, 1.4] rețeaua generează valorile [1.5, 7.7, 2.3]. Aceasta înseamnă că rețeaua consideră că planta cu cele patru caracteristici menționate aparține celei de-a doua clase, cea cu label-ul 1 (clasele au label-urile 0, 1, 2). Dacă acest lucru corespunde cu valoarea label-ului din setul de date de antrenare, înseamnă că rețeaua a clasificat corect datele de intrare, prin urmare acest rezultat contribuie la acuratețea rețelei. După ce se parcurge setul de date de antrenare, se determină acuratețea procentuală a rețelei (nr datelor clasificare corect / numărul tuturor datelor).

### 2. Realizați o evaluare a rețelei neuronale

Împărțiți setul de date în două părți: 33% din date vor constitui setul de antrenare, restul vor constitui setul de test. Antrenați rețeaua folosind setul de antrenare și determinați acuratețea acesteia folosind setul de test. Afișați acuratețea la sfârșitul fiecărei epoci (Similar cu cerința 1.)

3. Citiți de la tastatură un set de valori ale caracteristicilor plantelor și, folosind rețeaua antrenată, afișați probabilitățile ca planta cu setul de valori introdus de utilizator să aparțină celor trei specii. Probabilitățile de apartenență la clasele vizate se pot obține aplicând funcția *F.softmax()* pe valorile de ieșire ale rețelei (valorile de ieșire = cele care se obțin cu metoda *forward()*)

4. Determinați numărul optimal de epoci de antrenare. Evaluați rețeaua pentru un număr de epoci din mulțimea {2, ... , nr\_max\_epoci} și determinați numărul de epoci pentru care acuratețea este suficient de mare (de exemplu, câte epoci sunt necesare pentru ca acuratețea > 0.95)

5. Realizați o validare încrucișată (cross-validation) a rețelei. Folosiți 10% din datele disponibile pentru antrenarea rețelei, și evaluați rețeaua folosind restul de 90%. Apoi alegeți alte 10% instanțe ale datelor, antrenați din nou rețeaua și evaluați folosind restul datelor. Repetați până când întregul set de date s-a utilizat pentru antrenare și apoi afișați acuratețea medie ce rezultă în urma etapelor de antrenare-testare.

## Clasificarea imaginilor folosind rețele neuronale convoluționale

Rețelele neuronale convoluționale (Convolutional Neural Networks, prescurtat CNN) sunt clasificatori special concepuți pentru lucrul cu imagini. Structura acestora facilitează clasificarea datelor cu un număr mare de parametri, deoarece aceste tipuri de rețele permit prelucrarea eficientă a acestor categorii de date.

CNN nu au o arhitectură sau o structură prestabilită, aceasta se adaptează la tipul de imagine care se prelucrează sau la natura obiectelor care trebuie clasificate. Fig 1. ilustrează un exemplu de astfel de arhitectură.

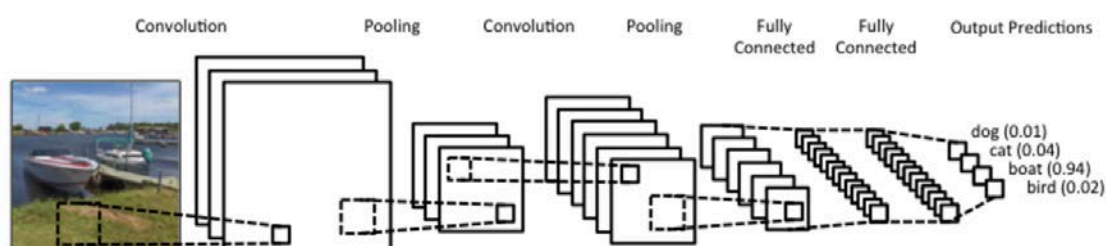


Fig. 1. Exemplu de CNN. La intrare se trimite o imagine, care este supusă operațiilor corespunzătoare straturilor de convoluție (*convolution*) și agregare (*pooling*). Rezultatul se propagă prin straturi complet conectate (*fully connected*). Ieșirea rețelei constă într-un vector de probabilități ce încadrează conținutul imaginii într-o mulțime de clase preexistente.

În general, CNN au două componente:

- I. O componentă de extragere a trăsăturilor din imagini. Aceasta se compune din mai multe straturi, care pot fi:
  - Straturi de convoluție – în cadrul acestora se realizează convoluția dintre imaginea de intrare și un filtru (matrice  $n \times n$  de dimensiuni mici). Spre deosebire de straturile din rețelele neuronale clasice, în straturile de convoluție numărul de conexiuni se reduce la dimensiunea filtrelor utilizate, și nu la cea a imaginii de la intrare. Straturile de convoluție produc una sau mai multe *feature maps*, imagini care conțin anumite trăsături sau caracteristici ale imaginii de la intrare (Fig 2). Valorile filtrelor cu care se face convoluția în cadrul unui anumit strat constituie ponderile conexiunilor din acel strat. La antrenarea rețelei, aceste ponderi se modifică după o logică similară cu cea a rețelelor clasice, ideea fiind de a minimiza eroarea dintre rezultatul de la ieșire dorit și cel obținut efectiv.

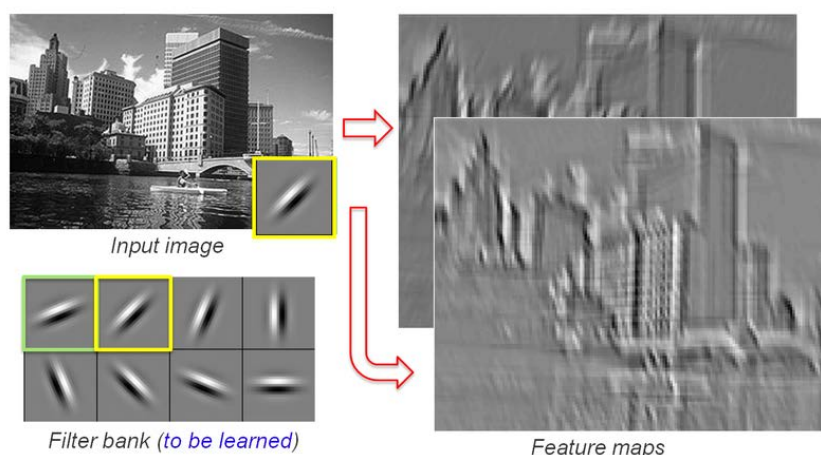


Fig. 2. Imaginea de la intrare și feature map-urile care rezultă în urma aplicării convoluției cu anumite filtre

- Straturi de agregare (*engl: pooling*) – în cadrul acestor straturi se realizează o reducere a dimensiunii feature map-urilor de la intrare. Astfel, operația de agregare 2x2 aplicată pe un feature map de dimensiune 64x64 generează un feature map de dimensiune 32x32. Reducerea dimensiunii pentru fiecare grup 2x2 de pixeli din feature map-ul de la intrare se poate realiza prin mai multe metode: de ex. Se determină maximul dintre cei 2x2 pixeli (*max pooling*), se poate face suma lor (*sum pooling*) sau se poate determina valoarea lor medie (*average pooling*). Fig 3. ilustrează rezultatul operațiilor *max pooling* și *sum pooling*.

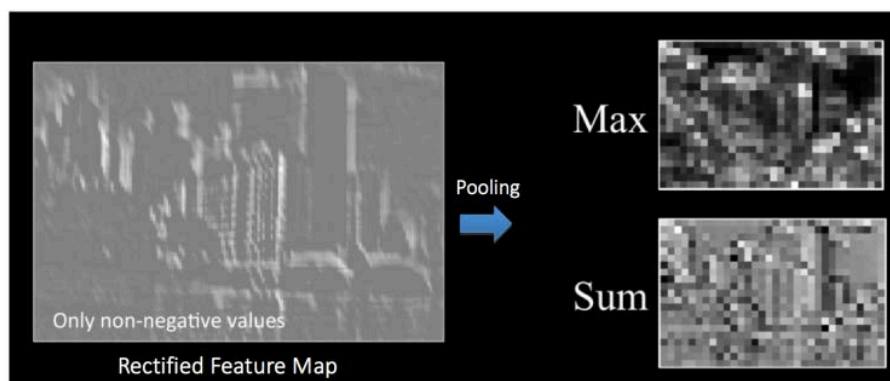


Fig. 3. Rezultatul operației de agregare, care constă în reducerea dimensiunii unui feature map.

Straturile de convoluție și agregare sunt supuse unei funcții de activare cu rolul de a asigura comportamentul neliniar al rețelei. Ca funcție de activare, de cele mai multe ori se folosește ReLU (Rectified Linear Unit). Fig. 4 ilustrează rezultatul aplicării funcției ReLU pe un feature map.

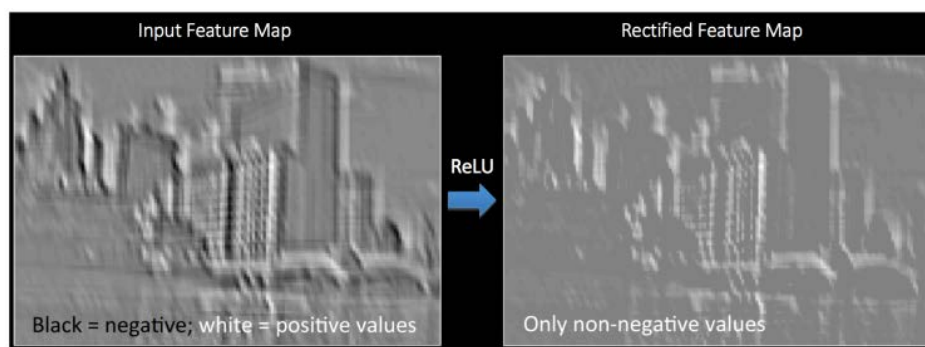


Fig. 4. Rezultatul operației de agregare, care constă în reducerea dimensiunii unui feature map.

- II. O componentă complet conectată (*fully connected*), în cadrul căreia se realizează clasificarea propriu-zisă. Această componentă este o rețea neuronală clasică, la intrarea căreia se furnizează feature map-urile, și la ieșirea căreia se aplică funcția de activare *softmax*. Ieșirea acestei componente este un vector de probabilități cu un număr de componente egal cu numărul de clase. Fiecare componentă a vectorului reprezintă probabilitatea ca imaginea de la intrare să se încadreze în clasa corespunzătoare.

Exemplificăm aspectele menționate până acum în Fig. 5.

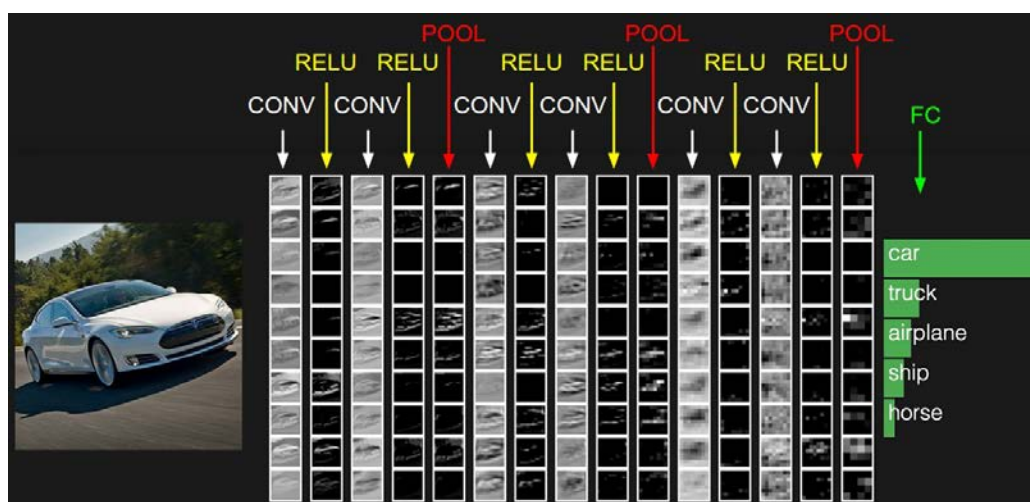


Fig. 5. Rezultatele generate de o rețea neuronală de convoluție. Se observă multiple straturi de convoluție și agregare, precum și etapele în care se aplică funcția de activare ReLU. Imaginile intermediare sunt feature map-urile generate în cadrul fiecărui pas.

Imaginea de la intrare conține un obiect care trebuie încadrat într-una din cele 5 clase preexistente (categoriile vizibile în extremitatea dreaptă a imaginii din Fig. 5). Imaginea este supusă mai multor operații de convoluție (CONV), agregare (POOL) și rectificare (RELU), de fiecare dată generându-se mai multe feature map-uri. Acestea sunt imagii care conțin trăsături semnificative ale obiectelor din imaginea inițială. Trăsăturile sunt apoi supuse unui proces de



clasificare prin intermediul unei rețele neuronale complet conectate (FC), rezultând probabilitățile ca imaginea să aparțină fiecărei categorii.

### Aplicație:

Următoarea secvență de cod implementează o CNN cu un strat de convoluție, unul de agregare (*pooling*) și unul fully-connected. Scopul este de a proiecta o rețea care să recunoască cifre în imagini grayscale, folosind fișierele din setul de date din aplicația care însoțește documentația pentru a antrena rețeaua. Listăm codul care implementează rețeaua în întregime, apoi îl vom analiza pe porțiuni.

```
class SimpleCNN(nn.Module):
    def __init__(self, imgWidth, imgHeight):
        super(SimpleCNN, self).__init__()

        inputWidth = imgWidth
        inputHeight = imgHeight
        nrConvFilters = 3
        convFilterSize = 5
        poolSize = 2
        outputSize = 10

        self.convLayer = nn.Conv2d(1, nrConvFilters, convFilterSize)
        self.poolLayer = nn.MaxPool2d(poolSize)
        fcInputSize = (inputWidth - 2*(convFilterSize // 2)) * (inputWidth -
2*(convFilterSize // 2)) * nrConvFilters // (2 * poolSize)
        self.fcLayer = nn.Linear(fcInputSize, outputSize)

    def forward(self, input):
        output = self.convLayer(input)
        output = self.poolLayer(output)
        output = F.relu(output)
        output = output.view([1, -1])
        output = self.fcLayer(output)
        return output

    def train(self, images, labels):
        lossFunc = nn.CrossEntropyLoss()
        nrEpochs = 10
        learnRate = 0.01
        optimizer = torch.optim.SGD(self.parameters(), learnRate)













        for epoch in range(nrEpochs):

            for image, label in zip(images, labels):

                optimizer.zero_grad()
                predicted = self.forward(image.unsqueeze(0))
                loss = lossFunc(predicted, label.unsqueeze(0))
                loss.backward()
                optimizer.step()

            print('Epoch', epoch, 'loss', loss.item())
```

Imaginile de antrenare constau în 1000 imagini 28x28 grayscale, câte 100 imagini pentru fiecare cifră. Clasa în care se încadrează fiecare imagine este reprezentată printr-un întreg din mulțimea  $\{0, \text{nrClase}-1\}$ . Pentru fiecare imagine, acest întreg se numește etichetă (*label*). În cazul de față, sunt 10 clase (cifrele '0', '1', '2', ..., '9') reprezentate prin label-urile 0, 1, 2, ..., 9. Este o coincidență faptul ca label-urile și denumirile claselor coincid. Aceeași metodă de codificare s-ar aplica pentru oricare alte clase, indiferent de semnificația lor (nu doar pentru cele care reprezintă cifre). Așadar, datele de antrenare sunt perechi formate din imagini și label-urile care indică cifra conținută:

Images	Labels
	0
	0
	0
...	...
	1
	1
	1
...	...
	2
	2
	2
...	...
...	...
...	...
	9
	9
	9
...	...

### Semnificația liniilor de cod este următoarea:

```
class SimpleCNN(nn.Module):
```

Rețeaua neuronală este organizată sub forma unui modul. Un modul în Pytorch implementează o serie de operații predefinite și poate la rândul său conține mai multe submodule.

```
def __init__(self, imgWidth, imgHeight):
```

Constructorul primește ca parametri dimensiunile unei imagini.

```
inputWidth = imgWidth  
inputHeight = imgHeight
```

Dimensiunile datelor de intrare. Este nevoie de acestea pentru a defini dimensiunea stratului fully-connected, cel convoluțional va ști să deducă dimensiunea din tensorii pe care îi primește ca input.

```
nrConvFilters = 3  
convFilterSize = 5  
poolSize = 2
```

Trei parametri care servesc la definirea straturilor speciale ale rețelelor convoluționale.

```
outputSize = 10
```

Dimensiunea stratului de ieșire al rețelei = numărul de clase

```
self.convLayer = nn.Conv2d(1, nrConvFilters, convFilterSize)
```

Stratul convoluțional: primul parametru se referă la numărul de valori ale pixelilor imaginilor de intrare (i.e. numărul de valori de culoare, *color channels*). Întrucât lucrăm cu imagini grayscale, acest parametru are valoarea 1. Stratul convoluțional generează 3 feature maps aplicând 3 filtre convoluționale de dimensiune 5x5 pe imaginile de intrare.

```
self.poolLayer = nn.MaxPool2d(poolSize)
```

Stratul de agregare: reduce dimensiunea feature map-urilor de intrare de 2 ori. **Max** se referă la faptul că reducerea se face alegând valoarea maximă din fiecare grup de pixeli ai feature map-urilor de intrare.

```
fcInputSize = (inputWidth - 2*(convFilterSize // 2)) * (inputHeight - 2*(convFilterSize // 2)) * nrConvFilters // (2 * poolSize)
```

```
self.fcLayer = nn.Linear(fcInputSize, outputSize)
```

Stratul fully connected: dimensiunea de intrare a acestui strat se determină pe baza rezultatelor operațiilor de convoluție și agregare anterioare, astfel:

Stratul convoluțional generează 3 feature map-uri folosind filtre 5x5. În urma aplicării filtrelor pe imaginea de la intrare de dimensiune 28x28, rezultă 3 feature maps de dimensiune 24x24. Se pierde câte 2 pixeli din fiecare extremitate a imaginii inițiale deoarece se realizează convoluția cu un filtru 5x5, fără padding sau alte operații suplimentare. Pe aceste filtre se aplică o operație de agregare 2x2, care reduce dimensiunea feature map-urilor de 2x pe fiecare dimensiune. Așadar, dimensiunea stratului fully connected va fi  $12 \times 12 \times 3 = 432$ . Dimensiunea datelor de la ieșire este egală cu nr. de clase (în cazul nostru, 10).

```
def forward(self, input):
    output = self.convLayer(input)
    output = self.poolLayer(output)
    output = F.relu(output)
    output = output.view([1, -1])
    output = self.fcLayer(output)
    return output
```

Metoda care realizează propagarea înainte. În ordine, se realizează următoarele operații:

- se propagă imaginea prin stratul de convoluție și se generează feature map-uri la ieșire.
- pe aceste feature map-uri se aplică o agregare 2x2. Fiecare grup de 2x2 valori din feature map-ul inițial se reduce la o singură valoare, egală cu maximumul celor patru.
- pe rezultatul anterior se aplică funcția de activare ReLU.
- se "aplatizează" rezultatul anterior, care dintr-un tensor de dimensiune [12,12,3] devine un tensor de dimensiune [1, 432].
- se propagă rezultatul anterior prin stratul fully connected
- valorile de ieșire constau într-o mulțime de 10 valori numerice (câte una pentru fiecare clasă) care, dacă ar fi normalizate, ar indica probabilitățile ca imaginea de la intrarea metodei **forward** să se încadreze în fiecare din cele 10 clase.

```
def train(self, images, labels):
```

Această metodă realizează antrenarea rețelei. Algoritmul de antrenare folosește imaginile de antrenare (images) pentru care se cunosc clasele în care se încadrează (labels).

```
lossFunc = nn.CrossEntropyLoss()
```

Funcția cu care se evaluează eroarea dintre valorile furnizate de rețeaua neuronală și labelurile din setul de date de antrenare.

```
nrEpochs = 10
```

Numărul de epoci – o epocă este o etapă din faza de antrenare a rețelei în cadrul căreia întregul set de date de antrenare a fost folosit pentru ajustarea parametrilor rețelei. Așadar, în faza de antrenare rețeaua "vede" datele de antrenare de **nrEpochs** ori.

```
learnRate = 0.01
```

Rata de învățare = parametru care controlează măsura în care se modifică ponderile rețelei în faza de antrenare. Cu cât `learnRate` este mai mare, cu atât parametrii rețelei variază mai multe de la o faza de antrenare la alta.

```
optimizer = torch.optim.SGD(self.parameters(), learnRate)
```

Metoda de optimizare este algoritmul de ajustare al ponderilor rețelei în urma apariției unei erori între valoarea furnizată de rețea și cea din setul de date. În cazul nostru, metoda folosită se numește Stochastic Gradient Descent (SGD).

```
for epoch in range(nrEpochs):
```

Antrenarea se realizează pe parcursul mai multor epoci.

```
or image, label in zip(images, labels):
```

La fiecare epocă, întregul set de date de antrenare va fi supus procesărilor din faza de antrenare a rețelei.

```
optimizer.zero_grad()
```

Optimizarea presupune căutarea valorilor ponderilor rețelei care minimizează eroarea generată de rețea. Pentru aceasta, metoda de căutare se folosește de variațiile erorii în raport cu ponderile rețelei, exprimate prin componentele gradientului erorii în raport cu ponderile. Aceste componente ale gradientului trebuie resetate la începutul procesului de re-ajustare a ponderilor.

```
predicted = self.forward(image.unsqueeze(0))
```

**predicted** este rezultatul generat de rețeaua neuronală pentru imaginea dată ca parametru de intrare. În cazul de față **predicted** este un tensor ce conține 10 valori, câte una pentru fiecare clasă. Cu cât valoarea aferentă unei clase este mai mare, cu atât se consideră ca rețeaua asociază acea clasă cu datele de intrare într-o măsură mai mare. Metoda **unsqueeze** adaugă o dimensiune suplimentară tensorului (necesară pentru aplicarea corectă a funcției de calcul al erorii)

```
loss = lossFunc(predicted, label.unsqueeze(0))
```

Se determină diferența dintre valoarea generată de rețea și label-ul din setul de date, folosind funcția definită anterior.

```
loss.backward()
```

```
optimizer.step()
```

Se realizează propagarea înapoi a erorii prin rețeaua neuronală și se ajustează ponderile rețelei în sensul minimizării erorii, cu metoda implementată în optimizer.

**Sarcini:**

1. Pentru fiecare epocă, determinați și afișați eroarea de clasificare a rețelei (nr de imagini clasificate greșit / nr de imagini din acea epocă)
2. Ajustați parametrii rețelei, prin încercări, astfel încât să obțineți o eroare cât mai mică, pentru un număr cât mai mic de epoci.
3. Realizați o evaluare a rețelei:
  - Împărțiți setul de imagini și labels în două părți. Folosiți prima parte pentru a antrena rețeaua, cealaltă va fi setul de date de test.
  - Determinați și afișați eroarea de clasificare în cele două situații: când aceasta se determină folosind datele de antrenare și când se determină folosind datele de test.
4. Implementați o rețea clasică (non-convoluțională, formată doar din straturi nn.Linear) și antrenați-o cu aceleași date (imaginile de această dată vor fi furnizate sub forma de vectori 1D, așadar primul strat din rețea va avea dimensiunea de input 28x28). Comparați convergența acestei rețele cu cea a rețelei convoluționale (i.e. câte epoci sunt necesare pentru ca eroarea să scadă sub o valoare prag)
5. Creați câteva imagini cu cifre și clasificați-le folosind rețeaua proiectată. Atenție, imaginile pe care le primește rețeaua trebuie să fie de aceleași dimensiuni ca și cele ale imaginilor folosite la antrenare (28x28 grayscale).

## Sisteme de recomandare

Sistemele de recomandare (*Recommender Systems* - RS) sunt algoritmi care realizează predicții pentru luarea de decizii prin prelucrări statistice ale caracteristicilor elementelor dintr-un sistem de tip utilizator-produs sau stare-acțiune. RS se utilizează cu precădere pentru identificarea celor mai bune sugestii privind produsele și serviciile oferite de o platformă (ex. magazin online), target-ul fiind comunitatea de utilizatori ai acelei platforme.

RS sunt de mai multe tipuri, funcție de prelucrările pe care le realizează. Cele mai multe astfel de sisteme sunt:

- Bazate pe conținut - RS realizează legături între o baza de date de utilizatori și o baza de date cu produse prin cuvinte cheie deduse pe baza caracteristicilor utilizatorilor și produselor. În acest sens, se iau în calcul atât specificațiile produselor, cât și detalii ce provin din profilurile utilizatorilor
- Colaborative – RS care realizează asocierea produs-utilizator pe baza popularității produselor. Aceste sisteme iau în calcul măsura în care produsele au fost selectate/cumpărate/evaluate de către utilizatorii unei comunități, pentru a face recomandări utilizatorilor care încă nu au întâlnit acele produse.

RS utilizate de către platformele de mari dimensiuni sunt sisteme hibride, constând într-un pipeline complex ce implică și prelucrare de conținut și procesare de tip colaborativ. În continuare vom studia RS ce presupun prelucrare de tip colaborativ, întrucât acestea apar cel mai frecvent în sistemele din lumea reală.

Adesea, RS colaborative primesc datele de intrare sub forma unei matrice de rating-uri, care are următoarele caracteristici:

- matricea are dimensiunile  $N \times M$ 
  - o N utilizatori
  - o M produse
- doar o submulțime a utilizatorilor au dat rating-uri unei submulțimi de produse. Perechile utilizator-produs pentru care există rating al produsului din partea utilizatorilor sunt în general mult mai mici decât  $M \times N$
- prin urmare, matricea de rating-uri este *rară* (engl. *sparse matrix*) (Fig 1.)
  - o elementele definite sunt în număr restrâns
  - o majoritatea elementelor sunt nedefinite


					
	5	3		3	
		1			5
	5	4	3		
	1	2			4
		1			
			4	5	

Fig. 1. Exemplu de matrice de rating-uri

Scopul unui RS colaborativ este ca, pornind de la valorile disponibile din matrice, să se determine estimări ale valorilor lipsă. Un exemplu în acest sens este ilustrat în Fig. 2, unde problema care se pune este: care dintre produsele pe care al doilea utilizator nu le-a evaluat încă i se pot recomanda acestuia? Decizia se ia prin realizarea de predicții (estimări) ale valorilor lipsă din linia corespunzătoare utilizatorului. Funcție de rating-urile estimate, se decide dacă un anumit produs i se poate recomanda acestuia. De exemplu, dacă ratingurile  $\in [1, 5]$  și valoarea estimată pe poziția (2, 1) este 4.5, atunci se poate decide faptul că primul produs constituie o recomandare adecvată pentru al doilea utilizator.

		Produse				
						
Utilizatori		5	3		3	
		?	1	?	?	5
		5	4	3		
		1	2			4
			1			
				4	5	

Fig. 2. Matrice de rating-uri unde problema care se pune este în ce măsură sunt recomandabile utilizatorului al doilea produsele pentru care acesta nu a dat rating



**Sarcini:****1. Fie următoarea matrice de ratinguri:**

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Să se determine rating-ul pe care utilizatorul **Alice** l-ar da produsului **Item5** prin următoarele metode:

**1.1. Colaborare bazată pe similaritatea dintre utilizatori**

Se estimează valoarea rating-ului lipsă pe baza similarității dintre Alice și ceilalți utilizatori, care au dat rating pentru Item5. Vom calcula similaritatea dintre doi utilizatori folosind **corelația Pearson**:

$$sim(a, b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}}$$

unde:

- $a, b$  – doi utilizatori
- $r_{a,p}$  – rating-ul dat de utilizatorul  $a$  obiectului  $p$
- $\bar{r}_a$  = rating-ul mediu dat de utilizatorul  $a$
- $P$  – mulțimea de obiecte care au primit rating-uri de la  $a$  și  $b$

Rating-ul lipsă se poate estima astfel:

$$pred(a, p) = \bar{r}_a + \frac{\sum_{b \in N} sim(a, b) * (r_{b,p} - \bar{r}_b)}{\sum_{b \in N} sim(a, b)}$$

Formula realizează o estimare a rating-ului pe care utilizatorul  $a$  l-ar da produsului  $p$ :

- $a$  – utilizatorul care nu a dat rating produsului  $p$
- $\bar{r}_a$  = rating-ul mediu dat de utilizatorul  $a$
- $b$  - unul dintre utilizatorii care au dat rating produsului  $p$
- $N$  – mulțimea utilizatorilor care au dat rating obiectului  $p$
- $r_{b,p}$  = rating-ul dat de utilizatorul  $b$  produsului  $p$
- $\bar{r}_b$  = rating-ul mediu dat de utilizatorul  $b$

## 1.2. Colaborare bazată pe similaritatea dintre produse

Se estimează valoarea rating-ului lipsă pe baza similarității dintre Item5 și celelalte produse, care au primit rating de la Alice. Vom calcula similaritatea dintre două produse folosind **similaritatea cosinus**:

$$\text{sim}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}$$

a, b sunt două produse. Se tratează a, b ca fiind vectori și se determină cosinusul unghiului format de cei doi. Cosinusul se calculează ca fiind raportul dintre produsul scalar al vectorilor și produsul lungimilor lor.

De exemplu, presupunem că a și b au primit ratingurile  $r_{a1}$ ,  $r_{a2}$ ,  $r_{b1}$ ,  $r_{b2}$  de la doi utilizatori. Atunci:

$$\vec{a} = [r_{a1} \ r_{a2}]$$

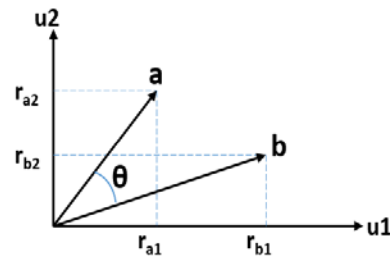
$$\vec{b} = [r_{b1} \ r_{b2}]$$

$$\text{sim}(\vec{a}, \vec{b}) = \cos(\theta)$$

$$\text{sim}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|}$$

$$\vec{a} \cdot \vec{b} = r_{a1} * r_{b1} + r_{a2} * r_{b2}$$

$$|\vec{a}| = \sqrt{r_{a1}^2 + r_{a2}^2}$$



Calcululele sunt similare în cazul în care produsele au rating-uri de la mai mulți utilizatori, singura diferență fiind faptul că vectorii au mai multe coordonate ( $r_{a1}$ ,  $r_{a2}$ ,  $r_{a3}$ , ...)

Rating-ul lipsă se poate estima astfel:

$$\text{pred}(u, p) = \frac{\sum_{i \in \text{ratedItem}(u)} \text{sim}(i, p) * r_{u,i}}{\sum_{i \in \text{ratedItem}(u)} \text{sim}(i, p)}$$

u – utilizatorul care nu a dat rating produsului p

i – produsele care au primit rating de la u

$r_{ui}$  – ratingul dat de u produsului i

## 2. Fie următoarea matrice de ratinguri:

	Item			
	W	X	Y	Z
User A		4.5	2.0	
User B	4.0		3.5	
User C		5.0		2.0
User D		3.5	4.0	1.0

Rating Matrix

Să se determine elementele lipsă (estimări ale rating-urilor utilizatorilor A, B, C, D pentru produsele W, X, Y, Z pe care nu le-au evaluat încă) prin **factorizarea matricei**.

Factorizarea presupune descompunerea matricei într-un produs de două matrice de dimensiuni mai mici. În cazul nostru, vom descompune matricea rating-urilor (4x4) într-un produs de două matrice de dimensiuni (4x2) și (2x4):

	Item			
	W	X	Y	Z
User A		4.5	2.0	
User B	4.0		3.5	
User C		5.0		2.0
User D		3.5	4.0	1.0

Rating Matrix

$$=$$

	W	X
User A		
User B		
User C		
User D		

User Matrix

$$\times$$

		W	X	Y	Z

Item Matrix

Trebuie să determinăm elementele matricelor User și Item matrix. Odată determinate, pentru a realiza estimarea elementului BX din matricea de ratinguri se face produsul liniei B din matricea User și coloanei X din matricea Item. Identificarea matricelor factori se realizează prin metode numerice. Vom folosi în acest scop metoda gradientului descendent.

### Notăm

- N = numărul de utilizatori
- M = numărul de produse
- n = indexul unui utilizator,  $n \in [0..N-1]$
- m = indexul unui produs,  $m \in [0..M-1]$
- K = numărul liniilor, respectiv al coloanelor din cele două matrice factor
- k = indexul liniilor, respectiv coloanelor din cele două matrice factor,  $k \in [0..K-1]$
- R = matricea ratingurilor, de dimensiuni NxM

- $U$  = matricea utilizatorilor, de dimensiuni  $N \times K$
- $V$  = matricea produselor, de dimensiuni  $K \times M$

Atunci descompunerea matricei arată astfel:

$$R_{N \times M} = U_{N \times K} \times V_{K \times M}$$

Se cunosc o parte din elementele din  $R$  (celelalte trebuie estimate). Scopul este determinarea elementelor din  $U$  și  $V$ , pe baza cărora se va face estimarea.

- se inițializează  $u_{nk}$ ,  $v_{km}$  cu valori aleatoare
- dorim să determinăm valorile  $u_{nk}$ ,  $v_{km}$  pentru care este minimă eroarea:

$$MSE = \frac{1}{2} (r_{nm} - \sum_k u_{nk} v_{km})^2$$

Pe parcursul mai multor iterații:

- se determină eroarea
- se determină gradientii erorii în raport cu  $u_{nk}$ ,  $v_{km}$ , care au următoarea expresie:

$$\frac{\partial MSE}{\partial u_{nk}} = (\sum_k u_{nk} v_{km} - r_{nm}) v_{km}$$

$$\frac{\partial MSE}{\partial v_{km}} = (\sum_k u_{nk} v_{km} - r_{nm}) u_{nk}$$

- se ajustează valorile  $u_{nk}$ ,  $v_{km}$  funcție de valorile gradientilor și ai ratei de învățare  $\alpha$ :

$$u_{nk} = u_{nk} - \alpha \frac{\partial MSE}{\partial u_{nk}}$$

$$v_{km} = v_{km} - \alpha \frac{\partial MSE}{\partial v_{km}}$$

- algoritmul se oprește fie după un număr limită de iterații, fie atunci când eroarea nu se mai modifică semnificativ de la o iterație la alta.

## Clasificare bazată pe trăsături

Metodele de recunoaștere a obiectelor din imagini se bazează pe identificarea unor trăsături care să permită clasificarea cu erori minime a acestora. În caz general, trăsăturile se reprezintă sub forma unui vector de valori reale deduse prin aplicarea unei multitudini de operații și transformări ale imaginilor sau ale unor regiuni din imagini.

Scopul identificării trăsăturilor este de a determina acele valori reale care sunt reprezentative pentru obiectele care se doresc identificate, și irelevante pentru restul informațiilor din imagini (background, zgomot etc.). Astfel, trăsăturile utile pentru soluționarea problemelor de clasificare:

- sunt similare pentru obiecte din imagini diferite care se încadrează în aceeași clasă
- diferă semnificativ dacă obiectele provenite din imagini diferite aparțin unor clase diferite

Principiul se ilustrează în Fig. 1. Trăsăturile utile asigură o separare clară a celor două clase reprezentate, în timp ce trăsăturile mai puțin utile îngreunează procesul de clasificare, deoarece clasele sunt mult mai “amestecate” în spațiul acelor trăsături.

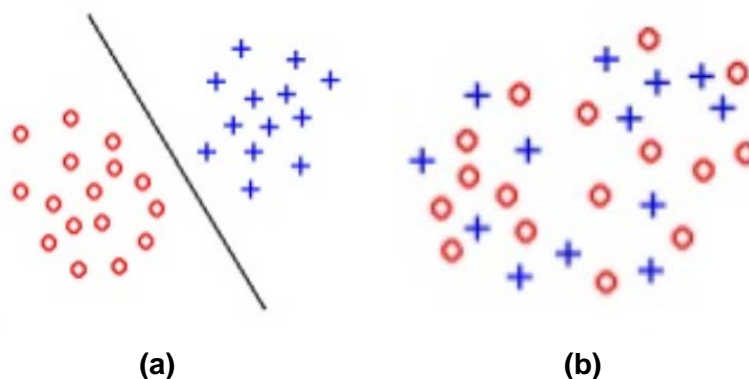


Fig. 1. Distribuția a două clase pe un set de date atunci când clasificarea se face folosind (a) trăsături reprezentative; (b) trăsături irelevante.

În cazul imaginilor, identificarea trăsăturilor este o problemă complexă, neexistând soluții universale valabile. Trăsăturile sunt dintr-o categorie foarte vastă și se pot deduce pe baza:

- distribuțiilor pixelilor din anumite regiuni ale imaginii
  - caracteristicilor contururilor / formelor / geometriei obiectelor
  - coeficienților obținuți prin aplicarea unor transformări globale asupra imaginii
- etc.

**Ca exemplu de determinare a trăsăturilor și de clasificare pe baza acestora, propunem următorul studiu de caz:**

Dorim să dezvoltăm un model de clasificare pentru recunoașterea amprentelor digitale. Datele disponibile constau într-o serie de imagini cu amprente și etichetele corespunzătoare lor. În cazul de față, prin “etichetă” se înțelege un identificator numeric al persoanei căreia îi aparține amprenta din imaginea corespunzătoare. Imaginile cu aceeași etichetă aparțin aceleiași clase, deoarece provin de la aceeași persoană. În Fig. 2 se prezintă câteva exemple de imagini din setul de date, alături de etichetele lor.



Fig. 2. Exemple de imagini cu amprente și etichetele corespunzătoare. Valorile etichetelor sunt id-uri ale persoanelor cărora le aparțin amprente.

Elaborarea unui model de clasificare pornind de la aceste date se referă la implementarea unei metode prin care să se realizeze asocieri între imagini și clasele corespunzătoare lor. În acest scop, în general nu se folosește întreaga informație prezentă în imagine, ci doar anumite detalii ale obiectelor conținute. Mulțimea acestor detalii formează vectorul de trăsături al imaginii corespunzătoare.

Pentru determinarea unor trăsături reprezentative, vom considera amprenta ca fiind compusă din creste (*ridges*) și văi (*valleys*) (Fig. 3). Pentru fiecare imagine, determinăm anumite puncte caracteristice ale amprentei (Fig. 4):

- punctele de la extremitățile creștelor (“capete”)
- punctele în poziția cărora creștele se bifurcă (“bifurcații”)



Fig. 3. Regiunile importante ale unei imagini cu amprente: crestele sunt regiunile negre, iar văile sunt porțiunile albe intermediare.

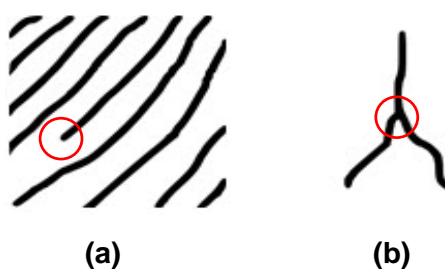


Fig. 4. Punctele caracteristice ale unei amprente: (a) capetele creștelor; (b) bifurcațiile.

Pentru determinarea punctelor caracteristice este necesară o preprocesare a imaginii, prin care să se scoată în evidență contururile creștelor. Aceasta etapă implică mai multe operații de filtrare a imaginii, de accentuare a contururilor creștelor și de subțiere a lor, rezultând contururi cu grosimea de 1 pixel. Un exemplu în acest sens se ilustrează în Fig. 4.



Fig. 4. Rezultatul preprocesării imaginilor: (a) imaginea inițială; (b) imaginea care rezultă prin evidențierea creștelor și subțierea acestora.

Pentru identificarea punctelor caracteristice, vom determina numărul tranzițiilor vecinilor pixelilor din imagine. Astfel, pentru fiecare pixel de pe creste:

- se consideră o vecinătate 3x3 în jurul pixelului (cei 8 vecini ai săi)
- se determină modulele diferențelor dintre fiecare doi vecini alăturați
- numărul tranzițiilor =  $0.5 * \text{suma modulelor}$

Dacă numărul tranzițiilor este:

- 1, atunci pixelul corespunzător este la capătul unei creste (Fig. 5(a))
- 2, atunci pixelul se află pe o creastă (dar nu la capătul acesteia) (Fig. 5(b))
- $\geq 3$ , atunci pixelul se află în poziția unei bifurcații (Fig. 5(c))

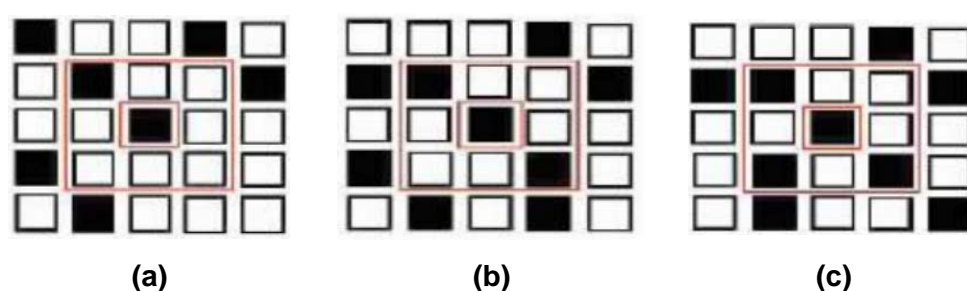


Fig. 5. Modalitatea de identificare a punctelor caracteristice: (a) capătul unei creste; (b) punct intermediar de pe o creastă; (c) punct aflat în poziția unei bifurcații.

În Fig 6 se prezintă o imagine unde sunt evidențiate punctele cheie. Pentru fiecare imagine vom determina vectorul de trăsături ca fiind  $[nc, nb]$ , unde  $nc$  = numărul capetelor, iar  $nb$  = numărul bifurcațiilor.

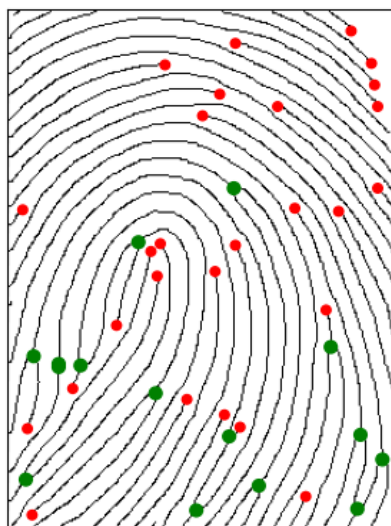


Fig. 6. Imagine în care sunt evidențiate **capetele** și **bifurcațiile**



**Sarcini:**

1. Determinați trăsăturile amprentelor din setul de imagini care însoțește documentația.
2. Aplicați metoda de clasificare K-Nearest Neighbors (vezi Lab 2) pe imaginile furnizate și determinați acuratețea acestora. Pentru a determina distanța dintre două imagini calculați distanța euclidiană dintre trăsăturile acestora.

## Generarea de secvențe folosind rețele neuronale recurente

Rețelele neuronale recurente (RNN) sunt special proiectate pentru prelucrarea datelor care formează secvențe. Diferența esențială dintre rețelele de acest tip și rețelele neuronale clasice o constituie straturile recurente, unde legăturile dintre neuroni sunt ciclice. RNN se utilizează în special pentru prelucrarea de secvențe de text și imagini, unde fiecare element al secvenței de cuvinte/caractere/imagini poate depinde de contextul creat de elementele anterioare ale secvenței.

Principiul de funcționare al RNN se bazează pe faptul că stările straturilor recurente se actualizează după fiecare element din secvență furnizat la intrarea rețelei. Astfel, presupunem că pentru o secvență de intrare  $x_0, x_1, \dots, x_t, \dots$ , se generează secvența de ieșire  $o_0, o_1, \dots, o_t, \dots$ . Starea stratului ascuns,  $s$ , se actualizează pentru fiecare pereche  $(x, o)$ , astfel încât starea  $s_t$  de la momentul  $t$  depinde de stările  $s_{t-1}, s_{t-2}, \dots$  de la momentele  $t-1, t-2, \dots$ . Așadar, valoarea de ieșire  $o_t$  depinde de prelucrările făcute în vederea generării valorilor  $o_{t-1}, o_{t-2}, \dots$ . În acest fel, la prelucrarea elementelor secvențelor se ține cont și de *contextul* creat de elementele anterioare ale secvenței. Spre exemplu, dacă se face prelucrarea unei secvențe de cuvinte, un anumit cuvânt dintr-o frază depinde de contextul creat de cuvintele anterioare din acea frază. Fraza însăși depinde de textul anterior. Pentru se conforma acestui principiu, starea rețelei RNN depinde de o parte sau de toate elementele secvenței, nu doar de cel din momentul actual, cum este cazul rețelelor non-recurente.

Structura unui strat recurent este ilustrată în Fig.1. La momentul  $t$ , pentru elementul  $x_t$  al secvenței, starea  $s_t$  depinde atât de  $x_t$ , cât și de starea  $s_{t-1}$ , cea determinată pentru elementul anterior  $x_{t-1}$  al secvenței.  $U$  și  $V$  sunt ponderile straturilor de intrare și ieșire, similare celor dintr-o rețea neuronală clasică, în timp ce  $W$  sunt ponderile ce asigură dependența între stările  $s$  de la momente diferite de timp.

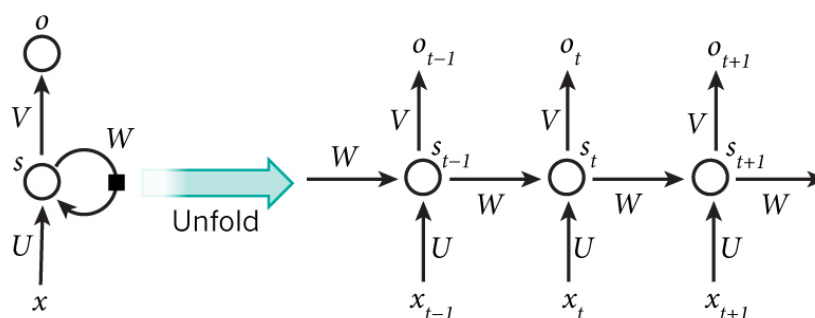


Fig. 1. Structura unei rețele neuronale recurente

O extensie a RNN sunt rețelele LSTM (*Long Short-Term Memory*), în cadrul cărora straturile recurente au o serie de componente suplimentare, numite *porți*. Acestea au rolul de a ameliora

problemele cauzate de gradientii cu valori prea mici sau prea mari (*vanishing gradients*, *exploding gradients*, vezi cursurile / documentația pentru detalii suplimentare).

Dorim să proiectăm o rețea recurentă pentru generarea de secvențe de text. Datele de antrenare provin dintr-un text oarecare, ideea fiind ca rețeaua să învețe limbajul folosit în text și să poată genera propriile secvențe folosind același limbaj.

Textul de antrenare este următorul:

*On a mango tree in a jungle, there lived many birds. They were happy in their small nests. Before the onset of the rainy season, all the animals of the jungle repaired their homes. The birds also made their homes more secure. Many birds brought twigs and leaves and others wove their nests. We should also store some food for our children, chirped one of the birds. And they collected food, until they had enough to see them through the rainy season. They kept themselves busy preparing for the tough times. Soon the rains came. It was followed by thunder and lighting. All the animals and birds stayed in their homes. It continued raining for many days. One day, a monkey went in the rain came into the forest. He sat on a branch, shivering with cold, Water dripping from its body. The poor monkey tried his best to get shelter, but in vain. The leaves were not enough to save him from the rains. It is so cold, said the monkey. The birds were watching all this. They felt sorry for the monkey but there was little they could do for him. One of them said, brother, Our small nests are not enough to give you shelter. Another bird said, all of us prepared for the rainy season. If you had, you would not be in this situation. How dare you tell me what to do, said the monkey, growling at the bird. The monkey angrily pounced on the birds nest, tore it and threw it on the ground. The bird and her chicks were helpless. The poor bird thought, fools never value good advice. It is better not to advise them.*

Principul de antrenare a rețelei este ilustrat în Figura 2:

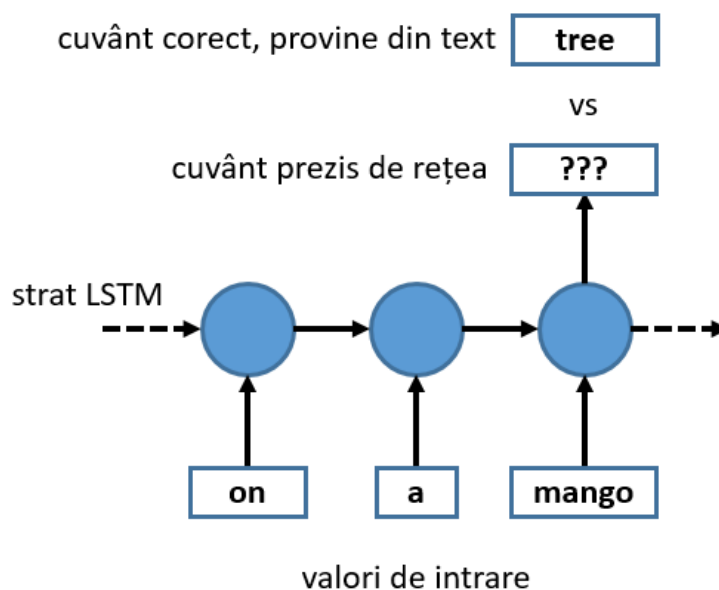


Fig. 2. Funcționarea rețelei LSTM în faza de antrenare. La intrare aceasta primește o secvență de trei cuvinte, iar la ieșire se compară cuvântul generat de rețea cu cel care urmează, preluat din textul folosit pentru antrenare.

Pentru antrenare trebuie generat **vocabularul** textului de antrenare - mulțimea cuvintelor/simbolurilor unice din text. Pentru textul de mai sus, vocabularul arată astfel:

```
[',', ' ', '.', 'a', 'advice', 'advise', 'all', 'also', 'and',
'angrily', 'animals', 'another', 'are', 'at', 'be', 'before', 'best',
'better', 'bird', 'birds', 'body', 'branch', 'brother', 'brought',
'busy', 'but', 'by', 'came', 'chicks', 'children', 'chirped', 'cold',
'collected', 'continued', 'could', 'dare', 'day', 'days', 'do',
'dripping', 'enough', 'felt', 'followed', 'food', 'fools', 'for',
'forest', 'from', 'get', 'give', 'good', 'ground', 'growling', 'had',
'happy', 'he', 'helpless', 'her', 'him', 'his', 'homes', 'how', 'if',
'in', 'into', 'is', 'it', 'its', 'jungle', 'kept', 'leaves',
'lighting', 'little', 'lived', 'made', 'mango', 'many', 'me',
'monkey', 'more', 'nest', 'nests', 'never', 'not', 'of', 'on', 'one',
'onset', 'others', 'our', 'poor', 'pounced', 'prepared', 'preparing',
'rain', 'raining', 'rains', 'rainy', 'repaired', 'said', 'sat',
'save', 'season', 'secure', 'see', 'shelter', 'shivering', 'should',
'situation', 'small', 'so', 'some', 'soon', 'sorry', 'stayed',
'store', 'tell', 'the', 'their', 'them', 'themselves', 'there',
'they', 'this', 'thought', 'threw', 'through', 'thunder', 'times',
```

```
'to', 'tore', 'tough', 'tree', 'tried', 'twigs', 'until', 'us',  
'vain', 'value', 'was', 'watching', 'water', 'we', 'went', 'were',  
'what', 'with', 'would', 'wove', 'you']
```

Vocabularul se poate genera folosind o listă / array / dicționar etc. Fiecărui cuvânt unic îi va corespunde un index numeric (numit și etichetă, *label*). Rețeaua neuronală prelucrează cuvintele folosind label-urile lor (primește la intrare o secvență de label-uri, la ieșire furnizează un label sau o altă secvență de label-uri).

Pentru detalii și exemple privind o astfel de rețea, consultați ultima parte din *Cursul 7*.

Implementarea rețelei descrise mai sus arată astfel (după codul sursă de mai jos urmează și explicațiile aferente):

```
textFile = open("rnn_text.txt")  
text = textFile.read()  
#separate punctuation by spaces  
punctuation = [',', '.', ':', ';', '?', '!', '"', "'"]  
tempCharList = [' ' + c if c in punctuation else c for c in text]  
text = ''.join(tempCharList)  
text = text.lower()  
  
#build vocabulary  
words = text.split()  
vocabulary = list(set(words))  
vocabulary.sort()  
  
#labels of words from training text:  
wordLabels = [vocabulary.index(w) for w in words]  
  
#build training data  
sequenceLength = 3  
noSequences = 100  
#random indices from training text:  
indices = random.sample(range(len(words)-sequenceLength-1), noSequences)  
  
inputs = [wordLabels[i : i+sequenceLength] for i in indices]  
targets = [wordLabels[i+sequenceLength] for i in indices]  
  
inputs = tr.tensor(inputs, dtype=tr.float)  
targets = tr.tensor(targets, dtype=tr.long)  
  
class SimpleRNN(nn.Module):  
    def __init__(self, inputSize, outputSize, lstmLayerSize, noLSTMLayers):  
        super(SimpleRNN, self).__init__()  
        self.inputSize = inputSize  
        self.lstmLayerSize = lstmLayerSize  
        self.outputSize = outputSize  
        self.noLSTMLayers = noLSTMLayers  
  
        self.lstmLayer = nn.LSTM(self.inputSize, self.lstmLayerSize,  
self.noLSTMLayers)  
        self.outLayer = nn.Linear(self.lstmLayerSize, self.outputSize)
```

```

def forward(self, input):
    input = input.view(-1, 1, 1)
    lstmOut, hidden = self.lstmLayer(input)
    outLayerInput = lstmOut[-1, 0, :]
    predictedOut = self.outLayer(outLayerInput)
    return predictedOut

def train(self, inputs, targets):
    noEpochs = 100
    learnRate = 0.001
    optimizer = tr.optim.Adam(self.parameters(), learnRate)
    lossFunc = nn.CrossEntropyLoss()

    for epoch in range(noEpochs):
        for input, target in zip(inputs, targets):
            optimizer.zero_grad()
            predicted = self.forward(input)
            loss = lossFunc(predicted.unsqueeze(0), target.unsqueeze(0))
            loss.backward()
            optimizer.step()

        print('Epoch', epoch, 'loss', loss.item())

myRNN = SimpleRNN(1, len(vocabulary), 16, 1)
myRNN.train(inputs, targets)

def generateText(desiredTextLength):
    while True:
        sentence = input('Introduceti %s cuvinte sau _quit: ' % sequenceLength)
        sentence.strip()
        if sentence == "_quit":
            break
        words = sentence.split()
        if len(words) != sequenceLength:
            continue

        try:
            inputLabels = [vocabulary.index(w) for w in words]
        except:
            print('Cuvintele introduse trebuie sa faca parte din vocabular.')
            continue

        sentence += ' '
        for i in range(desiredTextLength - sequenceLength):
            rnnInput = tr.tensor(inputLabels, dtype=tr.float)
            rnnOut = myRNN(rnnInput)
            outputLabel = rnnOut.argmax().item()
            outputWord = vocabulary[outputLabel]
            sentence += outputWord
            sentence += ' '
        print(sentence)

```

## Explicații:

```

textFile = open("rnn_text.txt")
text = textFile.read()
#separate punctuation by spaces
punctuation = [',', '.', ':', ';', '?', '!', '"', "'"]
tempCharList = [' ' + c if c in punctuation else c for c in text]
text = ''.join(tempCharList)
text = text.lower()

#build vocabulary
words = text.split()
vocabulary = list(set(words))
vocabulary.sort()

#labels of words from training text:
wordLabels = [vocabulary.index(w) for w in words]

```

Se citește fișierul care conține textul de antrenare. Se separă cuvintele și semnele de punctuație și se generează un vocabular sub forma unei liste. Fiecare cuvânt sau semn de punctuație din text va avea asociat un index în vocabular (poziția cuvântului în vocabular). Termenul utilizat pentru acești indecși este *label*. De exemplu cuvântul 'advice' se află în vocabular la indexul 3, prin urmare label-ul său este 3. Rețeaua neuronală nu prelucrează în mod direct șiruri de caractere, ci secvențe formate din label-urile cuvintelor corespunzătoare.

```

#labels of words from training text:
wordLabels = [vocabulary.index(w) for w in words]

#build training data
sequenceLength = 3
noSequences = 100
#random indices from training text:
indices = random.sample(range(len(words)-sequenceLength-1), noSequences)

inputs = [wordLabels[i : i+sequenceLength] for i in indices]
targets = [wordLabels[i+sequenceLength] for i in indices]

inputs = tr.tensor(inputs, dtype=tr.float)
targets = tr.tensor(targets, dtype=tr.long)

```

Datele de antrenare constau în perechi formate din secvențe de 3 cuvinte – al patrulea cuvânt care urmează. De exemplu, în textul de antrenare, secvenței de 3 cuvinte din `inputs[i]` îi urmează cuvântul din `targets[i]`. Se generează 100 de astfel de secvențe. Rețeaua se va antrena folosind aceste secvențe, ideea fiind ca aceasta să asocieze oricare trei cuvinte consecutive cu al patrulea imediat următor.

```

class SimpleRNN(nn.Module):
    def __init__(self, inputSize, outputSize, lstmLayerSize, noLSTMLayers):
        super(SimpleRNN, self).__init__()
        self.inputSize = inputSize
        self.lstmLayerSize = lstmLayerSize
        self.outputSize = outputSize
        self.noLSTMLayers = noLSTMLayers

        self.lstmLayer = nn.LSTM(self.inputSize, self.lstmLayerSize,
self.noLSTMLayers)
        self.outLayer = nn.Linear(self.lstmLayerSize, self.outputSize)

```

Rețeaua neuronală este implementată prin clasa SimpleRNN. Semnificația diversilor parametri este următoarea:

**inputSize** = dimensiunea unui element din secvență. În cazul de față, un element este un simplu număr întreg (label-ul unui cuvânt), așadar în exemplu nostru acest parametru va avea valoarea 1. În caz general, un element al unei secvențe poate avea mai multe caracteristici (de ex. pixelii unei imagini dintr-o secvență video).

**lstmLayerSize** și **noLSTMLayers** – caracteristici ale straturilor recurente din rețeaua neuronală. Pentru detalii, consultați cursurile / documentația aferentă.

**outputSize** – dimensiunea output-ului rețelei neuronale este dimensiunea vocabularului. Pentru fiecare cuvânt/semn de punctuație, rețeaua neuronală generează o valoare reală care indică probabilitatea ca în secvență să urmeze acel cuvânt.

**lstmLayer** – obiect care implementează unul sau mai multe straturi de tip *Long Short-Term Memory* (i.e. straturile recurente ale rețelei).

**outLayer** – stratul de ieșire al rețelei, este un strat “clasic”, similar celor din rețelele neuronale non-recurente.

```

def forward(self, input):
    input = input.view(-1, 1, 1)
    lstmOut, hidden = self.lstmLayer(input)
    outLayerInput = lstmOut[-1, 0, :]
    predictedOut = self.outLayer(outLayerInput)
    return predictedOut

```

Metoda primește ca parametru o secvență de trei cuvinte și returnează un vector de dimensiunea vocabularului – câte o valoare reală pentru fiecare cuvânt posibil. Astfel, pentru fiecare secvență de trei cuvinte, rețeaua îl prezice pe al patrulea, ca fiind cel de pe poziția ce corespunde valorii maxime a vectorului returnat de această metodă.



```
def train(self, inputs, targets):
    noEpochs = 100
    learnRate = 0.001
    optimizer = tr.optim.Adam(self.parameters(), learnRate)
    lossFunc = nn.CrossEntropyLoss()

    for epoch in range(noEpochs):
        for input, target in zip(inputs, targets):
            optimizer.zero_grad()
            predicted = self.forward(input)
            loss = lossFunc(predicted.unsqueeze(0), target.unsqueeze(0))
            loss.backward()
            optimizer.step()

        print('Epoch', epoch, 'loss', loss.item())
```

Antrenarea este foarte asemănătoare cu cea a rețelelor studiate la laboratoarele precedente. În cadrul unei iterații, rețeaua primește o secvență de trei cuvinte consecutive la intrare (*input*), și pe al patrulea care urmează, la ieșire (*target*). Al patrulea cuvând prezis de rețea se regăsește în *predicted*, iar diferența dintre cuvântul prezis de rețea (*predicted*) și cel corect (*target*) se determină folosind funcția *lossFunc*. Scopul antrenării este ajustarea ponderilor rețelei astfel încât să se minimizeze această diferență. Minimizarea se realizează printr-o metodă de optimizare, implementată de către obiectul *optimizer*. Secvențele utilizate pentru antrenare sunt prelucrate de către rețea de multiple ori. Ori de câte ori în cadrul procesului de antrenare s-a parcurs întregul set de date, se spune ca mai trecut o *epocă*.

```
def generateText(desiredTextLength):
    . . .
```

Funcția generează o secvență de text folosind rețeaua neuronală descrisă anterior. Parametrul metodei este numărul de cuvinte al secvenței dorite. Pentru detalii privind modul de funcționare și exemple de execuție, consultați materialele de la cursul corespunzător.

## Sarcini

1. Determinați acuratețea rețelei la sfârșitul fiecărei epoci de antrenare (asemănător cu modul de calcul al acurateții de la laboratoarele anterioare)
2. Pentru textul furnizat ca exemplu și/sau propriile texte, ajustați parametrii rețelei astfel încât să obțineți o acuratețe cât mai mare (preferabil > 95%). Cu cât acuratețea de la sfârșitul antrenării este mai mare, cu atât textele generate de rețea ar trebui să fie mai corecte și mai coerente. De exemplu, se poate ajusta:

- numărul de epoci
- numărul datelor de antrenare (numărul secvențelor generate pornind de la textul de antrenare)
- dimensiunea secvențelor de cuvinte de la intrare (în exemplul nostru lungimea unei astfel de secvențe este 3)
- numărul de straturi LSTM și dimensiunea lor
- rata de învățare
- metoda de optimizare (clasa obiectului *optimizer*). Printre cei mai frecvent utilizați algoritmi menționăm SGD, Adam, RMSprop. Mai multe detalii se găsesc la următorul link:  
<https://pytorch.org/docs/stable/optim.html>

3. Antrenați și testați rețeaua folosind și alte secvențe de text (nu foarte lungi, cât mai simple, cu un vocabular de dimensiuni reduse)

**Observație:** Exemplul prezentat în cadrul laboratorului este simplu, iar textul folosit pentru antrenare este de mici dimensiuni, pentru ca antrenarea rețelei să se poată realiza într-un timp decent. Din acest motiv, chiar și pentru o rețea de mare acuratețe, este puțin probabil ca secvențele de text generate să fie foarte coerente și corecte dpdv gramatical. Pentru a genera text corect și coerent în limba engleză, este nevoie de date de antrenare mult mai mari, și de o rețea cu mult mai mulți parametri, ceea ce înseamnă timpi de antrenare de ordinul orelor, chiar zilelor. Pornind de la exemplul furnizat în laborator, se pot crea modele generative mult mai complexe și convingătoare. La următorul link se prezintă o aplicație “din lumea reală” cu rețele LSTM:

<https://arstechnica.com/gaming/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/>