

Horia-Alexandru MODRAN

ÎNVĂȚARE AUTOMATĂ (MACHINE LEARNING)



Editura
Universității
Transilvania
din Brașov

2026

EDITURA UNIVERSITĂȚII TRANSILVANIA DIN BRAȘOV

Adresa: Str. Iuliu Maniu nr.

500091 Brașov

Tel.: 0268 476 050

Fax: 0268 476 051

E-mail: editura@unitbv.ro

Editură recunoscută CNCSIS, cod 81

ISBN 978-606-19-1876-8 (e-book)

Copyright © Autorul, 2026

Lucrarea a fost avizată de Consiliul Departamentului de Electronică și Calculatoare, Facultatea de Inginerie electrică și știința calculatoarelor a Universității Transilvania din Brașov.

Cuprins

CUPRINS.....	3
PREFAȚĂ.....	5
1. CAPITOLUL 1. INTRODUCERE ÎN DATA MINING.....	7
1.1. Evoluția Tehnologiilor de date.....	7
1.2. Obiecte, atribute și tipuri de date	10
1.3. Utilizări ale Data Mining.....	16
1.4. Fluxul Data Mining	19
1.5. Analiza exploratorie a datelor	24
2. CAPITOLUL 2. FUNDAMENTELE ÎNVĂȚĂRII AUTOMATE	30
2.1. Inteligența Artificială și Învățarea Automată.....	30
2.2. Date și funcții în învățarea automată.....	33
2.3. Învățarea supervizată: clasificare și regresie.....	35
2.4. Învățarea nesupervizată și alte paradigme de învățare	40
2.5. Generalizare	48
3. CAPITOLUL 3. TEHNICI DE ÎNVĂȚARE SUPERVIZATĂ: CLASIFICARE.....	51
3.1. Clasificarea ca problemă de învățare supervizată.....	51
3.2. Arbori de decizie	54
3.3. Clasificare bazată pe instanțe și algoritmul KNN	61
3.4. Support Vector Machine (SVM).....	65
3.5. Evaluarea modelelor de clasificare.....	71
4. CAPITOLUL 4. TEHNICI DE ÎNVĂȚARE SUPERVIZATĂ: REGRESIE.....	74
4.1. Introducere în Regresie	74
4.2. Regresie liniară simplă și multiplă.....	78
4.3. Funcții de cost și evaluarea modelelor de regresie.....	86
4.4. Regresia Logistică	89

5.	CAPITOLUL 5. TEHNICI DE ÎNVĂȚARE NESUPERVIZATĂ: CLUSTERING.....	95
5.1.	Introducere în învățarea nesupervizată.....	95
5.2.	Conceptul de clustering și măsuri de proximitate.....	96
5.3.	Clustering Ierarhic.....	98
5.4.	Algoritmul K-Means.....	103
5.5.	Algoritmul DBSCAN.....	110
6.	CAPITOLUL 6. INTRODUCERE ÎN REȚELE NEURONALE.	115
6.1.	Inspirația biologică a Rețelelor Neuronale Artificiale	115
6.2.	Structura unei rețele neuronale artificiale	117
6.3.	Perceptronul și limitele de decizie.....	123
6.4.	Topologiile Rețelelor Neuronale	128
6.5.	Mecanismul de învățare și algoritmul de backpropagation.....	131
6.6.	Exemplu practic	136
7.	CAPITOLUL 7. REȚELE NEURONALE CONVOLUȚIONALE	143
7.1.	Reprezentarea caracteristicilor și operația de convoluție	143
7.2.	Parametrii operației de convoluție	146
7.3.	Pooling și reducerea dimensiunii datelor	149
7.4.	Arhitectura unei Rețele Neuronale Convoluționale	151
7.5.	Studii de caz : Evoluția Arhitecturilor CNN	154
8.	CAPITOLUL 8. PRELUCRAREA LIMBAJULUI NATURAL. ARHITECTURI TRANSFORMER.....	163
8.1.	NLP și pre-procesarea textului	163
8.2.	Reprezentarea textului: de la Bag-of-words la embeddinguri.....	168
8.3.	Metode de clasificare a textului	172
8.4.	Arhitecturi Transformers și mecanismul de auto-atenție	173
8.5.	Aplicații NLP moderne	176
	BIBLIOGRAFIE.....	186

Prefață

Lucrarea de față a fost concepută ca suport didactic pentru disciplinele pe care subsemnatul le susține în cadrul mai multor programe de masterat, având ca obiectiv introducerea studenților în domeniul actual și dinamic al învățării automate. Cartea se adresează în special studenților care urmează disciplinele „Analiza și prelucrarea datelor prin tehnici de Învățare Automată”, din cadrul programului de master Sisteme Electronice și de Comunicații Integrate, și „Tehnici de prelucrare a datelor și învățare automată aplicate în securitate cibernetică”, din cadrul programului de master Securitate Cibernetică, ambele organizate în cadrul Facultății de Inginerie Electrică și Știința Calculatoarelor – IESC. De asemenea, lucrarea este utilă pentru disciplina „Machine Learning”, din cadrul programului de master Tehnologii Mobile și Internet în E-Business, organizat de Facultatea de Matematică și Informatică, precum și pentru disciplina „Ingineria viitorului. Utilizarea AI în rezolvarea problemelor reale”, din cadrul programului de master Sisteme Mecatronice pentru Industrie și Medicină.

Necesitatea unei astfel de lucrări este determinată de rolul tot mai important al datelor, algoritmilor și modelelor inteligente în domeniile tehnice și informatice actuale. În prezent, capacitatea de a colecta, analiza, interpreta și valorifica datele reprezintă o competență esențială pentru viitorii specialiști.

Cartea pornește de la noțiuni introductive și nu presupune cunoștințe avansate anterioare în domeniul Inteligenței Artificiale sau al învățării automate, fiind necesare doar cunoștințe de bază de matematică și informatică. Conținutul este construit progresiv, de la conceptele fundamentale ale analizei datelor și Data Mining, până la metode moderne de învățare automată și învățare profundă.

Un aspect important al lucrării este îmbinarea fundamentelor teoretice cu exemple și situații practice. Înțelegerea unui algoritm nu înseamnă doar cunoașterea modului în care funcționează matematic, ci și capacitatea de a identifica situațiile în care acesta poate fi utilizat, de a pregăti corespunzător datele și de a interpreta rezultatele obținute.

Așadar, lucrarea urmărește dezvoltarea unor abilități foarte solicitate în prezent: înțelegerea fluxului complet al unui proiect de învățare automată, prelucrarea seturilor de date, alegerea algoritmilor adecvați, evaluarea modelelor și interpretarea limitărilor acestora.

Sper ca această carte să reprezinte un instrument util în formarea studenților și să le ofere acestora aptitudinile necesare pentru a putea înțelege, evalua și utiliza metode de învățare automată în probleme concrete.

1. Capitolul 1. Introducere în Data Mining

Dezvoltarea tehnologiilor informatice din ultimele decenii a determinat o creștere accelerată a volumului de date disponibile. Datele sunt generate astăzi de aplicații software, sisteme de comunicații, senzori, dispozitive mobile, platforme web, sisteme industriale și servicii digitale. În aceste condiții, simpla stocare a datelor nu mai este suficientă. Valoarea reală apare atunci când datele sunt analizate, interpretate și transformate în informații utile pentru înțelegerea fenomenelor, optimizarea proceselor și luarea deciziilor.

Data Mining reprezintă un domeniu aflat la intersecția dintre baze de date, statistică, învățare automată, algoritmi și calcul distribuit. Rolul său este de a oferi metode prin care colecțiile mari de date pot fi explorate automat sau semi-automat, în vederea identificării unor tipare, anomalii, corelații sau modele predictive. În contextul actual, Data Mining constituie una dintre bazele sistemelor moderne de învățare automată [7].

Capitolul introduce principalele concepte necesare pentru înțelegerea procesului de analiză a datelor. Sunt discutate noțiunile de obiect, atribut, tipuri de date, reprezentări numerice și categoricale, tehnici de transformare a datelor, utilizări ale Data Mining, precum și etapele generale ale unui flux de analiză: colectare, pre-procesare, extragere de informații și post-procesare. Capitolul se încheie cu elemente introductive de analiză exploratorie și vizualizare, deoarece acestea constituie punctul de plecare pentru orice proiect bazat pe date..

1.1. Evoluția Tehnologiilor de date

Primele sisteme informatice au fost orientate în principal spre colectarea și stocarea datelor. În anii 1960 și la începutul anilor 1970, prelucrarea fișierelor reprezenta principala modalitate de gestionare a informațiilor. Pe măsură ce volumul datelor a crescut, au apărut sistemele de gestiune a bazelor de date, care au permis organizarea informațiilor în structuri relaționale, interogarea eficientă și desfășurarea tranzacțiilor. Această etapă a creat infrastructura necesară pentru utilizarea sistematică a datelor în organizații și aplicații software.

Figura 1.1. evidențiază evoluția tehnologiilor de baze de date, pornind de la colectarea datelor și procesarea fișierelor, continuând cu sistemele de gestiune a bazelor de date și ajungând la analiza avansată a datelor, Data Warehouse și Data Mining. Diagrama arată că Data Mining nu apare izolat, ci ca rezultat al maturizării tehnologiilor de stocare, interogare și integrare a datelor. Pe măsură ce organizațiile au început să acumuleze volume mari de informații, a devenit necesară dezvoltarea unor tehnici capabile să extragă valoare din aceste colecții.

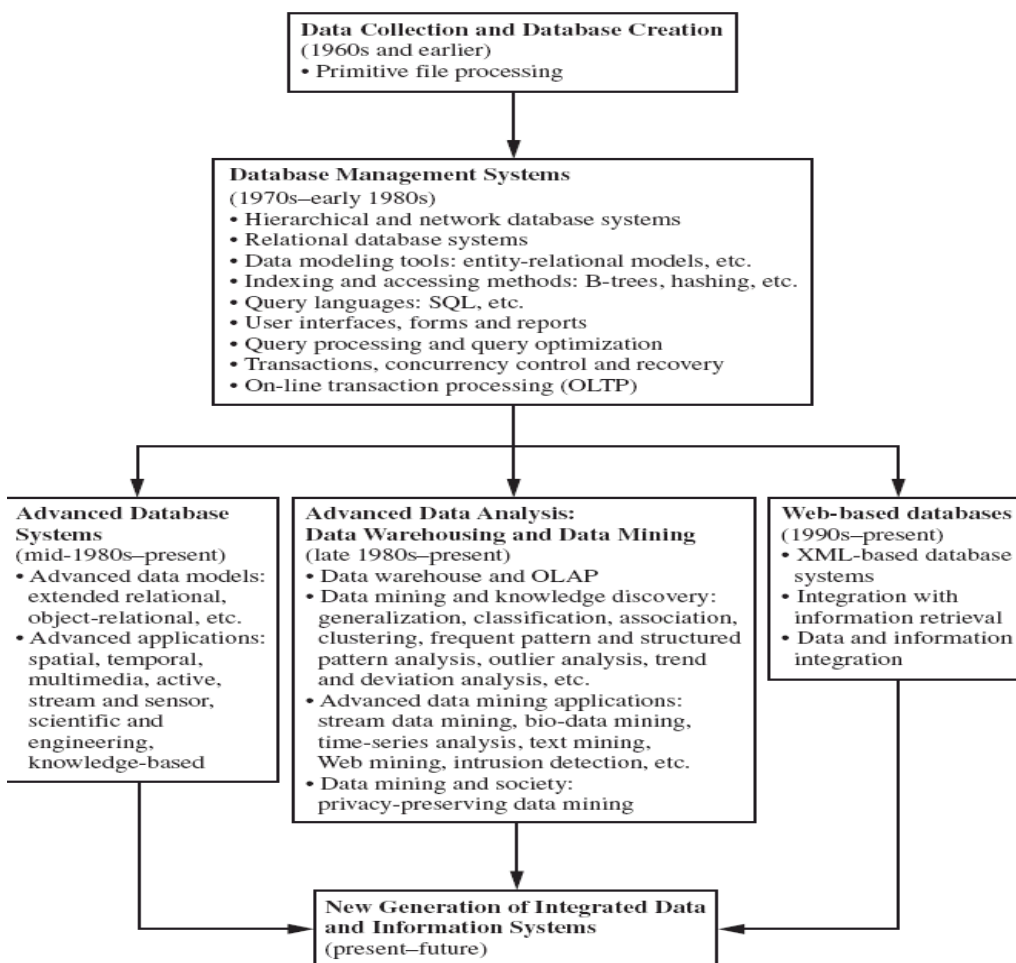


Fig. 1.1. *Evoluția tehnologiilor de baze de date și apariția Data Mining*

Termenul Data Mining a început să fie utilizat pe scară largă în anii 1990. Deși există mai multe definiții, ideea centrală este aceeași: **folosirea unor metode eficiente pentru**

analizarea colecțiilor mari de date și pentru descoperirea unor structuri utile, uneori neașteptate. Aceste structuri pot lua forma unor reguli, grupări, relații între variabile, modele predictive sau reprezentări simplificate ale datelor.

Figura 1.2 sintetizează foarte bine rolul domeniului: datele brute intră într-un proces de analiză, iar rezultatul urmărit este valoarea informațională. Această valoare poate însemna o predicție mai bună, o segmentare mai clară, identificarea unei anomalii, reducerea incertitudinii sau înțelegerea unui fenomen. Din acest punct de vedere, Data Mining poate fi privit ca o etapă de transformare: din volum mare de date spre cunoaștere utilizabilă.

Data Mining trebuie înțeles în relație cu domenii apropiate. Data Science se concentrează pe valorificarea datelor pentru înțelegerea și îmbunătățirea proceselor, Big Data vizează stocarea și procesarea unor volume foarte mari de date, iar Machine Learning și Deep Learning folosesc datele pentru construirea unor modele capabile să învețe. Între aceste domenii există suprapuneri, însă fiecare pune accentul pe o componentă diferită: infrastructură, analiză, modelare sau aplicabilitate.



Fig. 1.2. *Data Mining ca proces de transformare a datelor în valoare*

Figura 1.3 prezintă caracterul interdisciplinar al Data Mining. Baza de date asigură stocarea și accesul la informații, statistica oferă metode de descriere și inferență, algoritmi permit procesarea eficientă, vizualizarea ajută la interpretare, iar Machine Learning contribuie prin modele capabile să învețe din exemple. Calculul distribuit este important atunci când volumul de date depășește capacitățile unui sistem obișnuit.

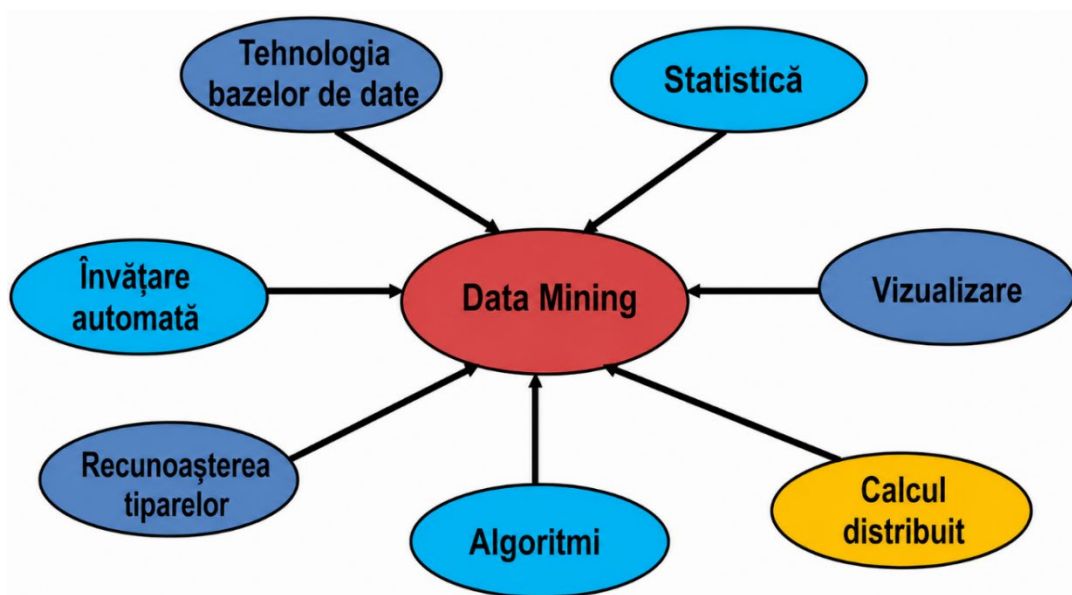


Fig. 1.3. Relația dintre Data Mining și domenii înrudite

Motivația principală pentru Data Mining este creșterea explozivă a volumului de date. În multe aplicații, datele sunt disponibile în cantități atât de mari încât analiza manuală devine imposibilă. Mai mult, datele sunt diverse: tabele relaționale, serii temporale, texte, imagini, grafuri, loguri de sistem, date de senzori sau interacțiuni ale utilizatorilor cu aplicații digitale. În aceste situații, algoritmi de Data Mining și Machine Learning oferă instrumente pentru identificarea automată a tiparelor relevante.

O observație importantă este că, în multe cazuri, datele în cantitate mare pot fi mai valoroase decât modelele foarte complexe aplicate pe seturi mici de date. Modelele moderne devin performante atunci când au la dispoziție suficiente exemple pentru a învăța tipare relevante. Acest principiu explică succesul unor aplicații precum corectarea automată a textelor, recomandările, clasificarea imaginilor sau analiza comportamentelor utilizatorilor.

1.2. Obiecte, atribute și tipuri de date

Înainte de a aplica metode de Data Mining, este necesar să înțelegem ce reprezintă datele și cum sunt organizate. În forma cea mai simplă, un set de date poate fi văzut ca o colecție de obiecte descrise prin atribute. Un obiect poate fi o persoană, un document,

o tranzacție, o imagine, un echipament, o conexiune de rețea sau orice altă entitate analizată. Atributele sunt proprietățile care descriu aceste obiecte, cum ar fi vârsta, temperatura, starea civilă, venitul, codul poștal, frecvența de apariție a unui cuvânt sau valoarea unui senzor.

Figura 1.4 ilustrează reprezentarea tabelară a unui set de date. Rândurile corespund obiectelor, iar coloanele corespund atributelor. Într-un astfel de tabel, fiecare înregistrare descrie o instanță a problemei analizate. De exemplu, într-un set de date despre contribuabili, atributele pot include rambursarea, starea civilă, venitul impozabil și o variabilă țintă. Dimensiunea setului de date este dată de numărul de obiecte, dimensionalitatea de numărul de atribute, iar sparsitatea de numărul valorilor efectiv populate în perechile obiect-atribut.

ID	Rambursare	Stare civilă	Venit impozabil	Fraudă
1	Da	Necăsătorit	125K	Nu
2	Nu	Căsătorit	100K	Nu
3	Nu	Necăsătorit	70K	Nu
4	Da	Căsătorit	120K	Nu
5	Nu	Divorțat	95K	NULL
6	Nu	Căsătorit	60K	Nu
7	Da	Divorțat	220K	Nu
8	Nu	NULL	85K	Da
9	Nu	Căsătorit	75K	Nu
10	Nu	Necăsătorit	90K	Da

Fig. 1.4. Reprezentarea datelor sub formă de obiecte și atribute

Atributele pot fi împărțite în mai multe categorii. Atributele categoriale descriu apartenența la o clasă sau categorie. Ele pot fi nominale, atunci când valorile nu au o ordine naturală, sau ordinale, atunci când valorile pot fi ordonate. De exemplu, culoarea

ochilor este un atribut nominal, în timp ce nivelul de înălțime exprimat prin scund, mediu și înalt poate fi considerat ordinal. Atributele numerice descriu cantități și pot fi discrete sau continue. Exemplele de atribute numerice continue includ temperatura, lungimea, ora, vârsta sau valoarea unui parametru măsurat.

În cazul datelor numerice, obiectele pot fi interpretate ca puncte într-un spațiu multidimensional. Fiecare atribut devine o dimensiune, iar fiecare obiect este descris prin coordonate în acel spațiu. Această interpretare este esențială pentru mulți algoritmi de Machine Learning, deoarece permite definirea noțiunilor de distanță, similaritate, vecinătate sau separare între grupuri de obiecte.

Figura 1.5 prezintă un exemplu simplu de date numerice, în care fiecare obiect este descris prin temperatură, umiditate și presiune. Matematic, setul de date poate fi reprezentat sub forma unei matrice cu n rânduri și d coloane, unde n este numărul de obiecte, iar d este numărul de atribute. Această reprezentare stă la baza multor operații de prelucrare, inclusiv normalizarea, calculul distanțelor și aplicarea algoritmilor de învățare automată.

	Temperatură	Umiditate	Presiune
O1	30	0.8	90
O2	32	0.5	80
O3	24	0.3	95



30	0.8	90
32	0.5	80
24	0.3	95

Fig. 1.5. Date numerice reprezentate printr-o matrice de date

Figura 1.6 arată modul în care datele numerice pot fi tratate ca puncte sau vectori. Pentru un număr mic de dimensiuni, aceste puncte pot fi reprezentate grafic, ceea ce permite observarea unor grupări sau separări. Pentru dimensiuni mai mari, interpretarea geometrică rămâne utilă, chiar dacă vizualizarea directă nu mai este posibilă. Această idee va fi folosită ulterior în algoritmi precum KNN, SVM sau K-Means.

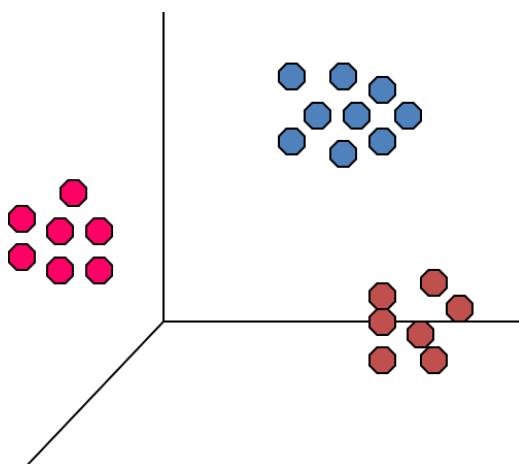


Fig. 1.6. Interpretarea geometrică a datelor numerice ca puncte sau vectori

În practică, multe seturi de date sunt relaționale mixte, adică includ atât atribute numerice, cât și atribute categoricale. De exemplu, într-un tabel pot apărea simultan vârsta, salariul, codul poștal, starea civilă și un atribut boolean. Unele valori pot fi reprezentate numeric, dar interpretate categorial. Codul poștal este un exemplu tipic: deși este reprezentat prin cifre, diferența numerică dintre două coduri poștale nu are aceeași semnificație ca diferența dintre două temperaturi.

Figura 1.7 ilustrează acest tip de date relaționale mixte. Codul poștal, starea civilă și treapta de venit sunt atribute categoricale, în timp ce vârsta și salariul sunt atribute numerice. În proiectele de învățare automată, interpretarea corectă a tipului de atribut este importantă, deoarece algoritmi pot produce rezultate greșite dacă un atribut categorial este tratat ca o valoare numerică obișnuită.

ia valori
numerice, dar
este de fapt
categoric

ID	Cod poștal	Vârstă	Stare civilă	Salariu	Treaptă venit	Restituire
1129842	45221	55	Necăsătorit	250000	Ridicat	0
2342345	45223	25	Căsătorit	30000	Scăzut	1
1234542	45221	45	Divorțat	200000	Ridicat	0
1243535	45224	43	Necăsătorit	150000	Mediu	0

Fig. 1.7. Exemplu de date relaționale mixte, cu atribute numerice și categoricale

Pentru ca algoritmi să poată procesa atributele categoriale, acestea sunt adesea transformate în reprezentări numerice. O abordare comună constă în introducerea unui atribut boolean pentru fiecare valoare posibilă a atributului categorial. Această tehnică este cunoscută în practică sub numele de codificare one-hot (*one-hot encoding*) și permite transformarea întregului set de date într-o matrice numerică.

Figura 1.8 prezintă transformarea unor categorii, cum ar fi codurile poștale și starea civilă, în coloane booleene. În loc să existe o singură coloană pentru starea civilă, apar coloane distincte pentru valorile posibile, cum ar fi Necăsătorit, Căsătorit sau Divorțat. Fiecare rând primește valoarea 1 pentru categoria prezentă și 0 pentru celelalte categorii. După această transformare, vectorul de atribute poate fi procesat numeric.

ID	Cod 45221	Cod 45223	Cod 45224	Vârstă	Necăsă- torit	Căsătorit	Divorțat	Venit	Rambursare
1129842	1	0	0	55	1	0	0	250000	0
2342345	0	1	0	25	0	1	0	30000	1
1234542	1	0	0	45	0	0	1	200000	0
1243535	0	0	1	43	1	0	0	150000	0

Fig. 1.8. Transformarea atributelor categoriale în atribute booleene

Operația inversă poate fi de asemenea utilă. Uneori atributele numerice sunt transformate în atribute categoriale prin gruparea valorilor în intervale. De exemplu, vârsta poate fi transformată în categorii precum 20 an (între 20 și 29), 40 ani (între 40 și 49) sau 50 ani (între 50 și 59), iar venitul poate fi transformat în trepte precum Scăzut, Mediu și Ridicat. Această transformare simplifică datele și poate face unele relații mai ușor de interpretat.

Figura 1.9 prezintă un exemplu în care valorile numerice ale vârstei și veniturii sunt grupate în categorii. Această discretizare poate fi utilă atunci când variațiile exacte ale valorilor nu sunt importante sau când se dorește obținerea unor reguli mai ușor de înțeles. Totuși, discretizarea implică și pierdere de informație, deoarece valorile apropiate, dar distincte, ajung să fie tratate identic.

Număr ID	Cod poștal	Vârstă	Stare civilă	Venit	Treaptă venit	Rambursare
1129842	45221	50 ani	Necăsătorit	Ridicat	Ridicat	0
2342345	45223	20 ani	Căsătorit	Scăzut	Scăzut	1
1234542	45221	40 ani	Divorțat	Ridicat	Ridicat	0
1243535	45224	40 ani	Necăsătorit	Mediu	Mediu	0

Fig. 1.9. Transformarea atributelor numerice în attribute categoriale prin intervale

Procesul de împărțire a unui interval numeric în subintervale poartă denumirea de *bucketization*. Există mai multe strategii. În cazul intervalelor de lățime egală, fiecare interval are aceeași dimensiune. În cazul intervalelor cu adâncime egală, fiecare interval conține aproximativ același număr de valori. O altă variantă este împărțirea logaritmică, utilă atunci când valorile acoperă ordine de mărime diferite.

Figura 1.10 compară trei modalități de discretizare în intervale. Marcajele albastre indică intervale de lățime egală, marcajele roșii corespund intervalelor cu același număr de puncte, iar marcajele verzi indică împărțirea logaritmică. Alegerea metodei depinde de distribuția datelor și de obiectivul analizei. Dacă datele sunt foarte concentrate într-o zonă, intervalele de lățime egală pot produce bin-uri aproape goale, în timp ce intervalele cu adâncime egală pot evidenția mai bine densitatea valorilor.



Albastru: Equi-width [20,40,60,80]

Roșu: Equi-depth

Verde: Equi-log ($\frac{end}{start} = 2$)

Fig. 1.10. Exemplu de împărțire a valorilor numerice în intervale prin bucketization

Pe lângă datele numerice și relaționale, Data Mining lucrează cu multe alte tipuri de date. Datele tranzacționale apar în aplicații precum analiza coșului de cumpărături sau

extragerea seturilor frecvente de itemi. Datele ordonate apar în serii temporale, loguri sau evoluția unor valori în timp. Datele spațiale includ poziții geografice, iar documentele pot fi reprezentate prin frecvența cuvintelor sau prin vectori semantici.

Figura 1.11 conține mai multe exemple: un tabel de tranzacții, o serie temporală, o hartă cu date spațiale și o matrice document-termeni. Aceste exemple arată diversitatea problemelor tratate în Data Mining. Deși datele au forme diferite, scopul rămâne același: identificarea unor structuri utile, relevante pentru problema analizată.

TID	Articole
1	Pâine, Coca-Cola, Lapte
2	Bere, Pâine
3	Bere, Coca-Cola, Lapte
4	Bere, Pâine, Lapte
5	Coca-Cola, Lapte, Miere

Date tranzacționale



Date ordonate

	team	coach	play	ball	score	game	wi	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

Date spațiale

Date documente

Fig. 1.11. Exemple de tipuri de date: tranzacționale, ordonate, spațiale și documente

1.3. Utilizări ale Data Mining

Data Mining poate fi privit din două perspective complementare. Din punct de vedere aplicativ, datele oferă un avantaj important atunci când pot fi transformate în informații utile. Din punct de vedere științific și tehnic, volumul mare de date disponibile impune dezvoltarea unor instrumente specifice pentru analiză, predicție, clasificare și descoperirea relațiilor ascunse. În ambele cazuri, accentul cade pe capacitatea de a exploata colecții mari și complexe de date.

Printre utilizările uzuale ale Data Mining se regăsesc descoperirea seturilor frecvente de itemi, extragerea regulilor de asociere, analiza exploratorie, măsurarea similarităților, gruparea datelor și clasificarea. Aceste sarcini sunt fundamentale și apar în numeroase aplicații practice: recomandarea de conținut, identificarea comportamentelor similare, detectarea anomaliilor, clasificarea documentelor, filtrarea mesajelor sau segmentarea utilizatorilor [13].

O primă sarcină importantă este analiza exploratorie a datelor. Aceasta urmărește înțelegerea setului de date înainte de aplicarea unui model. De exemplu, într-o platformă socială putem analiza cât de des postează utilizatorii, câte postări are fiecare utilizator, în ce momente apar postările, ce distribuții au valorile și dacă există corelații între numărul de postări și alte variabile. Alegerea metricilor potrivite este esențială, deoarece acestea determină ce aspecte ale datelor devin vizibile [23].

O altă sarcină fundamentală este explorarea similitudinilor. Multe aplicații presupun compararea obiectelor: doi utilizatori pot avea comportamente similare, două documente pot trata același subiect, două imagini pot conține obiecte asemănătoare sau două secvențe de evenimente pot indica același tip de activitate. Pentru a putea grupa sau recomanda elemente, trebuie mai întâi definită o măsură de similaritate.

Figura 1.12 prezintă un exemplu cu utilizatori descriși prin numărul de click-uri pe mai multe surse de conținut. Utilizatorii A, B și C par orientați spre conținut sportiv, în timp ce utilizatorii D, E și F par orientați spre surse de știri. O astfel de observație conduce natural la ideea de **clustering**: identificarea automată a unor grupuri de obiecte similare. Totuși, pentru a formaliza problema, trebuie stabilit cum se măsoară similaritatea dintre doi vectori de comportament.

	NBA	ESPN	Sports.com	MSNBC	NY Times	Wall Street	Politico
A	100	50	73	10	1	1	4
B	500	200	400	20	10	4	1
C	80	100	60	1	3	1	1
D	4	2	1	12	90	100	80
E	9	3	4	9	100	80	70
F	3	4	5	30	300	200	500

Fig. 1.12. Explorarea similitudinilor dintre utilizatori pe baza comportamentului de accesare

Predicția este o altă direcție esențială. Completarea unei valori lipsă poate fi privită ca o sarcină de predicție. Dacă se prezice o valoare continuă, problema este una de regresie. Dacă se prezice o valoare binară, problema este o clasificare binară. Dacă se prezice una dintre mai multe clase, problema este o clasificare multi-clasă. Exemplele includ predicția click-urilor pe reclame, predicția aprecierii unei postări, identificarea conținutului ofensator sau clasificarea automată a imaginilor.

Clasificarea are un rol central în învățarea automată. Procesul presupune identificarea caracteristicilor relevante, utilizarea unor exemple etichetate și dezvoltarea unui model capabil să prezică eticheta pentru instanțe noi. În multe aplicații moderne, clasificarea este componenta care permite sistemelor să ia decizii automate.

Clustering-ul reprezintă, la rândul său, o sarcină importantă. Fiind dat un set de puncte și o măsură de similaritate, obiectivul este identificarea unor grupuri astfel încât punctele din același grup să fie cât mai asemănătoare, iar punctele din grupuri diferite să fie cât mai diferite. Măsura de similaritate poate fi distanța euclidiană sau o măsură specifică problemei. În mod intuitiv, clustering-ul minimizează distanțele intracluster și maximizează distanțele dintre grupuri.

În ultimii ani, Deep Learning a devenit o direcție importantă a învățării automate. Modelele de învățare profundă folosesc rețele neuronale cu mai multe straturi, antrenate pe cantități mari de date. Aceste modele pot învăța reprezentări complexe și sunt utilizate în recomandări, analiză de text, recunoaștere de imagini și alte aplicații în care datele brute au structură complexă.

Figura 1.13 poziționează Deep Learning în interiorul domeniului Machine Learning, care la rândul său face parte din Inteligența Artificială. Ideea importantă este că învățarea profundă nu înlocuiește toate metodele Data Mining, ci completează instrumentele existente. În anumite probleme, modelele profunde reduc nevoia de extragere manuală a caracteristicilor, deoarece pot învăța automat reprezentări relevante din date.

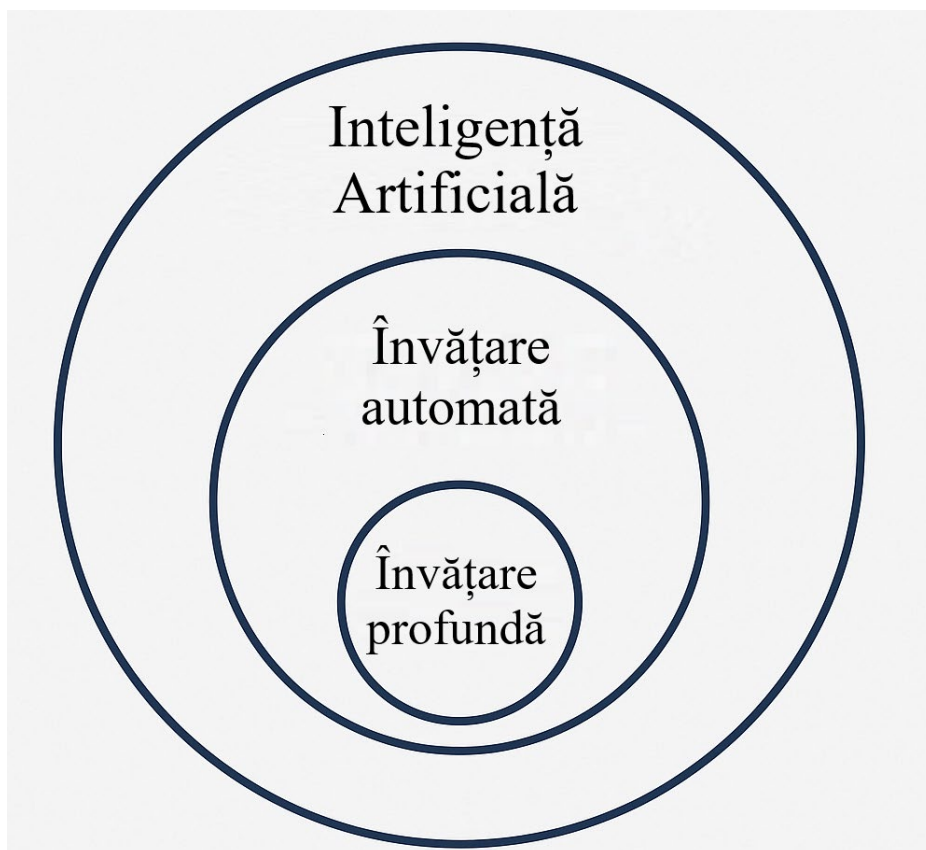


Fig. 1.13. Poziționarea învățării automate în cadrul IA

1.4. Fluxul Data Mining

Data Mining nu reprezintă întregul proces de analiză a datelor, ci doar una dintre componentele acestuia. Înainte ca algoritmi să poată extrage informații utile, datele trebuie colectate, curățate, transformate și pregătite. După aplicarea algoritmilor, rezultatele trebuie interpretate, vizualizate și validate. Din acest motiv, un proiect de învățare automată trebuie privit ca un flux complet, nu ca aplicarea izolată a unui algoritm.

Figura 1.14 sintetizează fluxul general: colectarea datelor, pre-procesarea, etapa de Data Mining și post-procesarea. Colectarea datelor furnizează materia primă. Pre-procesarea urmărește îmbunătățirea calității datelor și obținerea unei reprezentări adecvate. Etapa de Data Mining aplică metode analitice și algoritmi pentru extragerea informațiilor, iar post-procesarea transformă rezultatele în concluzii interpretabile.

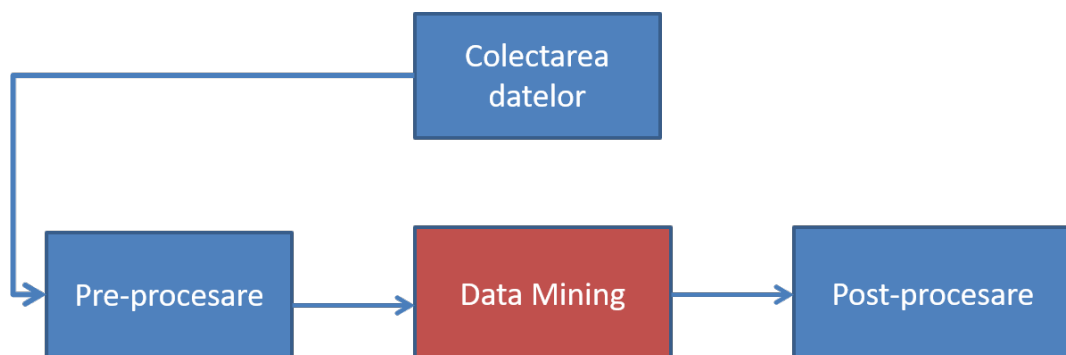


Fig. 1.14. Fluxul general al unui proces Data Mining

Colectarea datelor poate presupune accesarea unor baze de date, citirea unor fișiere, obținerea unor loguri, măsurători prin senzori sau preluarea unor fluxuri de informații. În multe situații, nu este eficient sau posibil să fie analizat întregul volum de date. Atunci se folosește eșantionarea, adică selectarea unui subset reprezentativ. Eșantionarea este folosită atât pentru analiza preliminară, cât și pentru analiza finală, mai ales atunci când datele sunt foarte numeroase.

Figura 1.15 evidențiază rolul dimensiunii eșantionului. Atunci când eșantionul este prea mic, anumite grupuri sau structuri pot lipsi complet din datele analizate. Pe măsură ce dimensiunea eșantionului crește, probabilitatea de a surprinde diversitatea setului de date crește. Totuși, o dimensiune mai mare implică și costuri mai ridicate de procesare.



Fig. 1.15. Influența dimensiunii eșantionului asupra reprezentativității datelor

După colectare, datele trebuie curățate. Problemele de calitate a datelor sunt foarte frecvente și pot afecta semnificativ rezultatele. Printre acestea se numără zgomotul, valorile extreme, valorile lipsă, datele duplicate și inconsistențele. Un model antrenat pe

date de calitate slabă va învăța tipare eronate sau va produce predicții instabile. De aceea, curățarea datelor este una dintre cele mai importante etape ale procesului.

Figura 1.16 prezintă un tabel în care apar mai multe probleme: o valoare extrem de mare a venitului, valori NULL și înregistrări duplicate sau inconsistente. Valoarea foarte mare poate fi o eroare de introducere sau poate reprezenta un caz real, deci nu trebuie eliminată automat fără analiză. Valorile lipsă trebuie tratate prin eliminare, imputare sau alte strategii, iar duplicatele trebuie identificate pentru a evita influențarea artificială a rezultatelor.

	ID	Rambursare	Stare civilă	Venit imposabil	Înșelăciune
	1	Da	Necăsătorit	125K	Nu
	2	Nu	Căsătorit	100K	Nu
	3	Nu	Necăsătorit	70K	Nu
	4	Da	Căsătorit	120K	Nu
Greșeală sau milionar?	5	Nu	Divorțat	10000K	Da
	6	Nu	NULL	60K	Nu
Valoare lipsă	7	Da	Divorțat	220K	NULL
	8	Nu	Necăsătorit	85K	Da
	9	Nu	Căsătorit	90K	Nu
Date inconsistente/duplicate	9	Nu	Necăsătorit	90K	Nu

Fig. 1.16. Probleme de calitate a datelor: valori extreme, valori lipsă și duplicate

O altă etapă importantă este extragerea caracteristicilor. Datele obținute nu sunt întotdeauna disponibile sub forma unui tabel relațional. Ele pot fi text, sunet, imagini, log-uri sau semnale. Pentru a aplica algoritmi de învățare automată, trebuie extrase caracteristici reprezentative. În cazul textului, caracteristicile pot fi cuvintele sau frecvențele acestora; în cazul imaginilor, pot fi muchii, texturi sau reprezentări învățate automat; în cazul logurilor, pot fi frecvențe, secvențe sau durate.

Normalizarea datelor este necesară atunci când atributele au scări diferite. Dacă un atribut are valori între 0 și 1, iar altul are valori între 0 și 100000, algoritmi bazați pe distanțe pot acorda o importanță disproporționată atributului cu valori mari.

Normalizarea coloanelor transformă fiecare atribut pe o scară comună, de exemplu în intervalul [0,1].

Figura 1.17 arată normalizarea coloanelor prin formula min-max. Pentru fiecare atribut se scade valoarea minimă și se împarte la diferența dintre maxim și minim. Rezultatul este un set de valori comparabile, fiecare atribut fiind exprimat în același interval. Această tehnică este utilă în special pentru algoritmi precum KNN, SVM sau K-Means.

Temperatură	Umiditate	Presiune		Temperatură	Umiditate	Presiune
30	0.8	90		0.75	1	0.66
32	0.5	80		1	0.40	0
24	0.3	95		0	0	1

Fig. 1.17. Normalizarea coloanelor prin scalarea valorilor în intervalul [0,1]

Normalizarea poate fi aplicată și pe rânduri. În analiza documentelor, de exemplu, numărul absolut de apariții ale unui cuvânt poate depinde de lungimea documentului. Prin împărțirea fiecărei valori la suma valorilor din rând, vectorul poate fi transformat într-o distribuție. Astfel, valorile exprimă proporții, nu frecvențe absolute.

Figura 1.18 prezintă normalizarea rândurilor pentru două documente descrise prin apariții de cuvinte. După normalizare, valorile pot fi interpretate ca probabilități relative. De exemplu, o anumită celulă exprimă probabilitatea ca un cuvânt ales aleatoriu din document să fie cuvântul corespunzător coloanei. Această transformare ajută la compararea documentelor de lungimi diferite.

	Cuvânt 1	Cuvânt 2	Cuvânt 3
Doc 1	28	50	22
Doc 2	12	25	13

	Cuvânt 1	Cuvânt 2	Cuvânt 3
Doc 1	0.28	0.5	0.22
Doc 2	0.24	0.5	0.26

Fig. 1.18. Normalizarea rândurilor pentru transformarea vectorilor în distribuții

O altă formă de normalizare pe rânduri se aplică în sistemele de recomandare. Doi utilizatori pot avea aceleași preferințe relative, chiar dacă unul oferă note mai mari în

general. Prin scăderea mediei fiecărui utilizator, se surprinde abaterea față de comportamentul mediu al acestuia, nu valoarea absolută a scorurilor.

Figura 1.19 arată că doi utilizatori pot evalua filmele în mod similar după ce se elimină diferența de nivel general al notelor. Înainte de normalizare, valorile sunt diferite; după scăderea mediei, structura preferințelor devine identică. Acest exemplu arată că pre-procesarea trebuie adaptată la semnificația datelor și la problema analizată.

	Film 1	Film 2	Film 3
Utilizator 1	1	2	3
Utilizator 2	2	3	4

	Film 1	Film 2	Film 3
Utilizator 1	-1	0	+1
Utilizator 2	-1	0	+1

Fig.1.19. Normalizarea rândurilor prin raportare la comportamentul mediu al utilizatorului

Post-procesarea rezultatelor urmărește interpretarea și prezentarea concluziilor. O componentă esențială este vizualizarea. O reprezentare grafică adecvată poate face vizibile tipare care nu sunt evidente într-un tabel. Histogramele, graficele de tip scatter plot, hărțile, graficele de evoluție și diagramele de distribuție sunt instrumente de bază pentru înțelegerea datelor. Ochiul uman este un instrument analitic puternic atunci când datele sunt reprezentate corespunzător. Vizualizarea nu înlocuiește algoritmi, dar ajută la verificarea ipotezelor, la identificarea valorilor extreme și la explicarea rezultatelor.

O provocare importantă este vizualizarea datelor multidimensionale. Oamenii pot interpreta ușor reprezentări în două dimensiuni și, într-o anumită măsură, în trei dimensiuni. Atunci când datele au zeci, sute sau mii de atribute, este necesară reducerea dimensionalității. Aceasta permite proiectarea datelor într-un spațiu mai mic, păstrând informația relevantă.

Figura 1.20 prezintă un exemplu intuitiv de reducere a dimensionalității. Deși setul inițial are șase dimensiuni, rândurile pot fi exprimate prin combinații ale unor vectori de bază. Astfel, datele pot fi rescrise într-o formă mai compactă. Reducerea dimensionalității este utilă atât pentru vizualizare, cât și pentru pre-procesare, deoarece reduce cantitatea de informație redundantă și poate îmbunătăți eficiența algoritmilor.

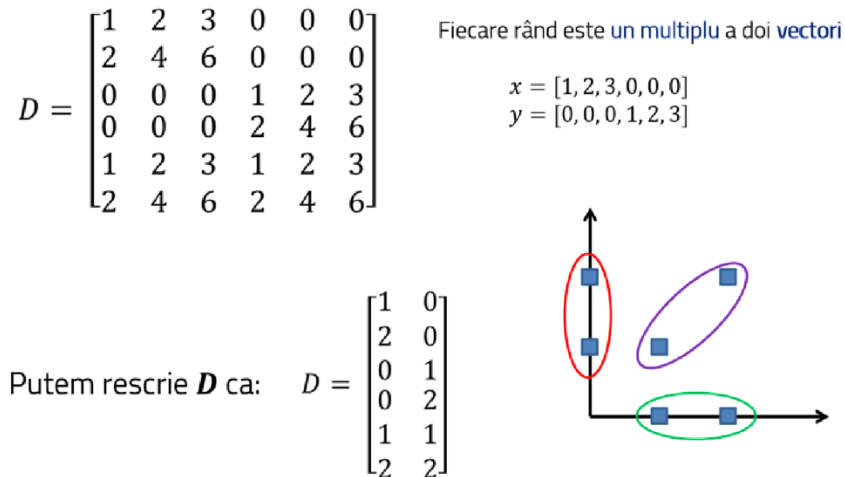


Fig. 1.20. Reducerea dimensionalității prin reprezentarea datelor într-un spațiu mai compact

1.5. Analiza exploratorie a datelor

Analiza exploratorie a datelor presupune folosirea unor metode simple, dar foarte importante, pentru a înțelege distribuțiile, valorile caracteristice și relațiile dintre atribute. Înainte de a construi un model, trebuie cunoscute proprietățile principale ale datelor: frecvența valorilor, mediana, răspândirea, valorile lipsă, valorile extreme și posibilele corelații.

Statisticile sumare sunt numere care descriu proprietăți ale datelor. Frecvența unei valori exprimă cât de des apare aceasta în setul de date. Modul este valoarea cea mai frecventă a unui atribut. Pentru atributele numerice, media, mediana, percentilele și abaterea standard sunt indicatori utilizați frecvent. Pentru atributele categoricale, frecvențele și diagramele asociate sunt adesea suficiente pentru o primă înțelegere.

Figura 1.21 prezintă un exemplu de analiză a atributelor Rambursare, Stare civilă și Venit. Valorile lipsă pot fi ignorate în anumite calcule, dar această decizie trebuie justificată. Pentru atributul Marital Status, modul este Single, deoarece apare cel mai frecvent. Histogramele și diagramele circulare permit observarea rapidă a distribuțiilor, iar tabelul rezumativ transformă datele brute în informație interpretabilă.

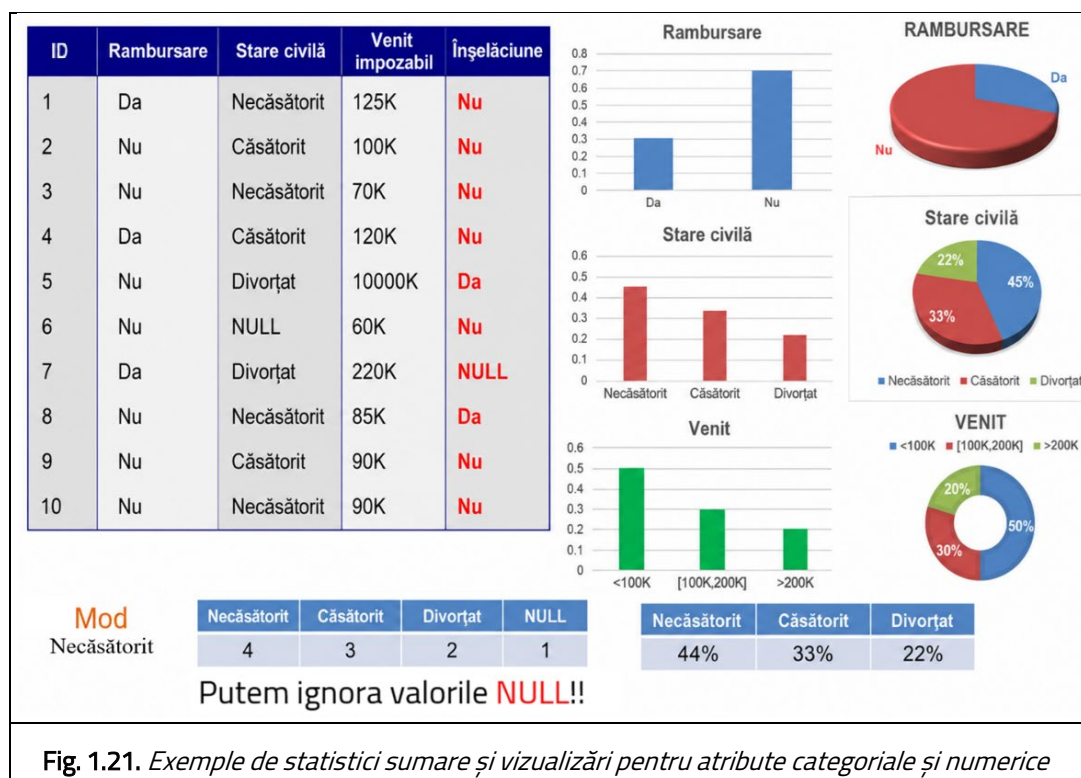


Fig. 1.21. Exemple de statistici sumare și vizualizări pentru atribute categoriale și numerice

Pentru date continue sau ordinale, percentilele sunt utile deoarece indică poziția unei valori în distribuție. Percentila p este valoarea sub care se află $p\%$ dintre observații. De exemplu, percentila 80 indică o valoare mai mare sau egală decât 80% dintre valorile observate. Percentilele sunt utile atunci când distribuțiile sunt asimetrice sau când apar valori extreme.

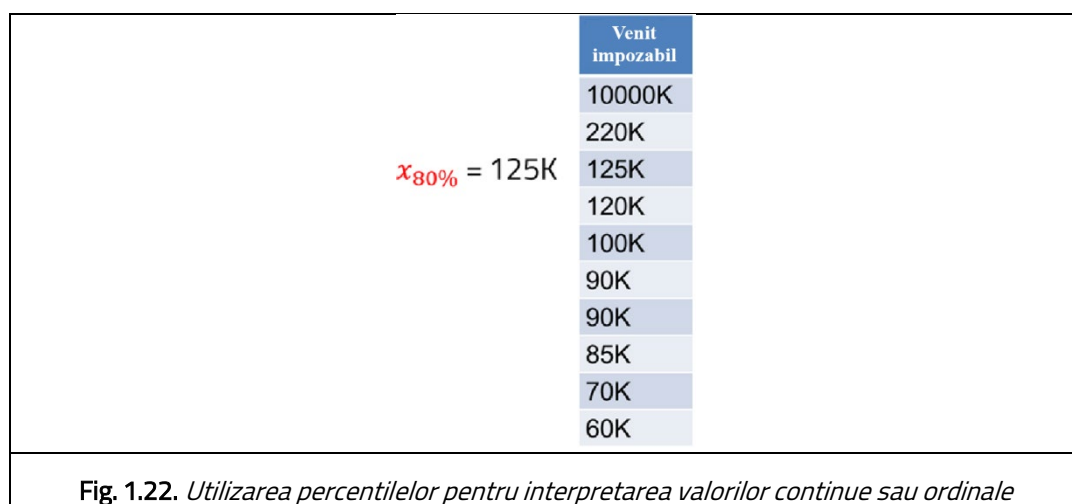


Fig. 1.22. Utilizarea percentilelor pentru interpretarea valorilor continue sau ordinale

Figura 1.23 arată diferența dintre medie, mediană și media trunchiată într-un set de date care conține o valoare extremă. Media devine 1096K, fiind distorsionată de valoarea 10000K. După eliminarea valorii minime și maxime, media trunchiată devine 112,5K, iar mediana este 95K. Exemplul arată de ce alegerea indicatorului statistic trebuie făcută în funcție de distribuția datelor.

$$\text{media}(x) = \bar{x} = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\text{mediana}(x) = \begin{cases} x_{(r+1)} & \text{dacă } m \text{ este impar, adică } m = 2r + 1 \\ \frac{1}{2}(x_{(r)} + x_{(r+1)}) & \text{dacă } m \text{ este par, adică } m = 2r \end{cases}$$

Fig. 1.23. Calcularea valorilor medii și mediane

În multe situații, nu este suficient să analizăm atributele separat. Este important să înțelegem relațiile dintre două sau mai multe atribute. Putem analiza, de exemplu, dacă starea civilă are legătură cu o anumită clasă, dacă rambursarea se corelează cu venitul sau dacă numărul anilor de studiu are legătură cu venitul. Astfel de analize sunt esențiale pentru construirea modelelor predictive.

Figura 1.24 prezintă două modalități de vizualizare a relației dintre starea civilă și variabila Cheat: matricea de confuzie și matricea de distribuție. Matricea de confuzie conține numărul de cazuri pentru fiecare combinație de valori, iar matricea de distribuție exprimă proporțiile corespunzătoare. Astfel de reprezentări ajută la identificarea unor dependențe între atribute categoriale.

Matricea de confuzie			Matricea de distribuție		
	Nu	Da		Nu	Da
Necăsătorit	3	1	Necăsătorit	0,3	0,1
Căsătorit	4	0	Căsătorit	0,4	0
Divorțat	1	1	Divorțat	0,1	0,1

Fig. 1.24. *Analiza relațiilor dintre atribute folosind matrici de confuzie și distribuție*

Pentru două atribute numerice, o reprezentare frecvent utilizată este graficul de tip scatter plot. Axa X reprezintă un atribut, iar axa Y îl reprezintă pe celălalt. Fiecare obiect devine un punct bidimensional. Dacă punctele urmează o direcție clară, se poate observa o corelație; dacă sunt răspândite aleatoriu, relația este mai slabă sau inexistentă.

Figura 1.25 prezintă un exemplu de scatter plot pentru relația dintre venit și anii de studiu. Prezența unei valori extreme poate distorsiona puternic percepția asupra relației. În astfel de situații, este necesară verificarea outlierilor și, uneori, folosirea unei scale diferite pentru a observa mai bine distribuția valorilor.

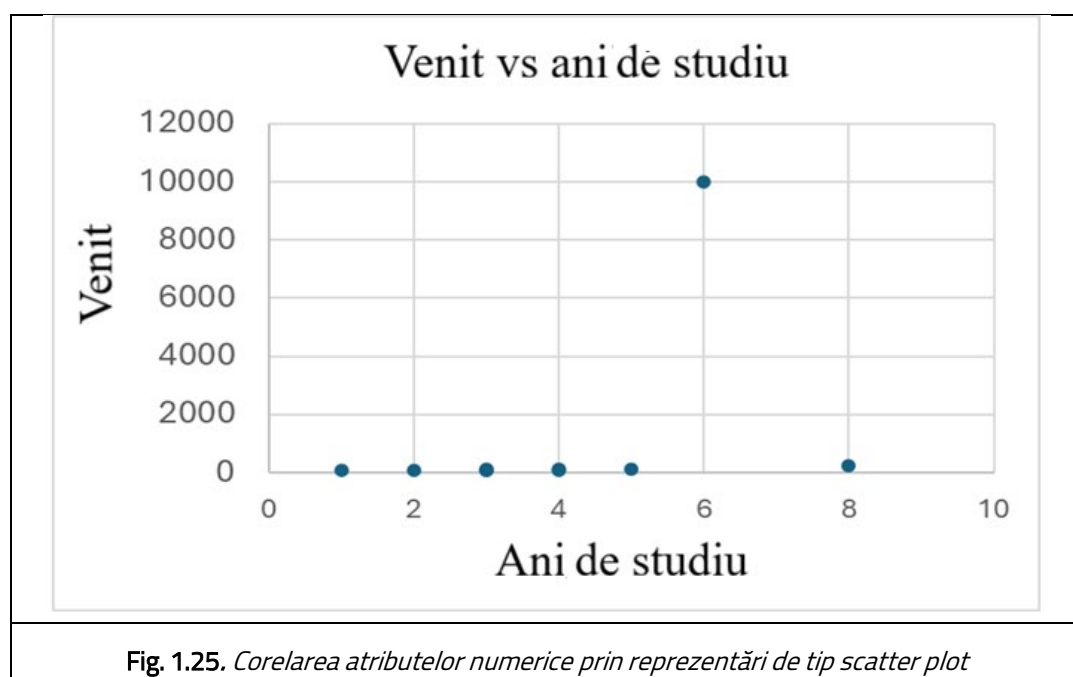


Figura 1.26 arată cum scara logaritmică pe axa Y poate face graficul mai ușor de interpretat atunci când valorile au ordine de mărime diferite. De asemenea, după eliminarea outlier-ului, relația dintre venit și anii de studiu devine mai clară. Această observație evidențiază importanța analizei exploratorii înainte de construirea unui model: un singur punct extrem poate schimba interpretarea întregului set de date.

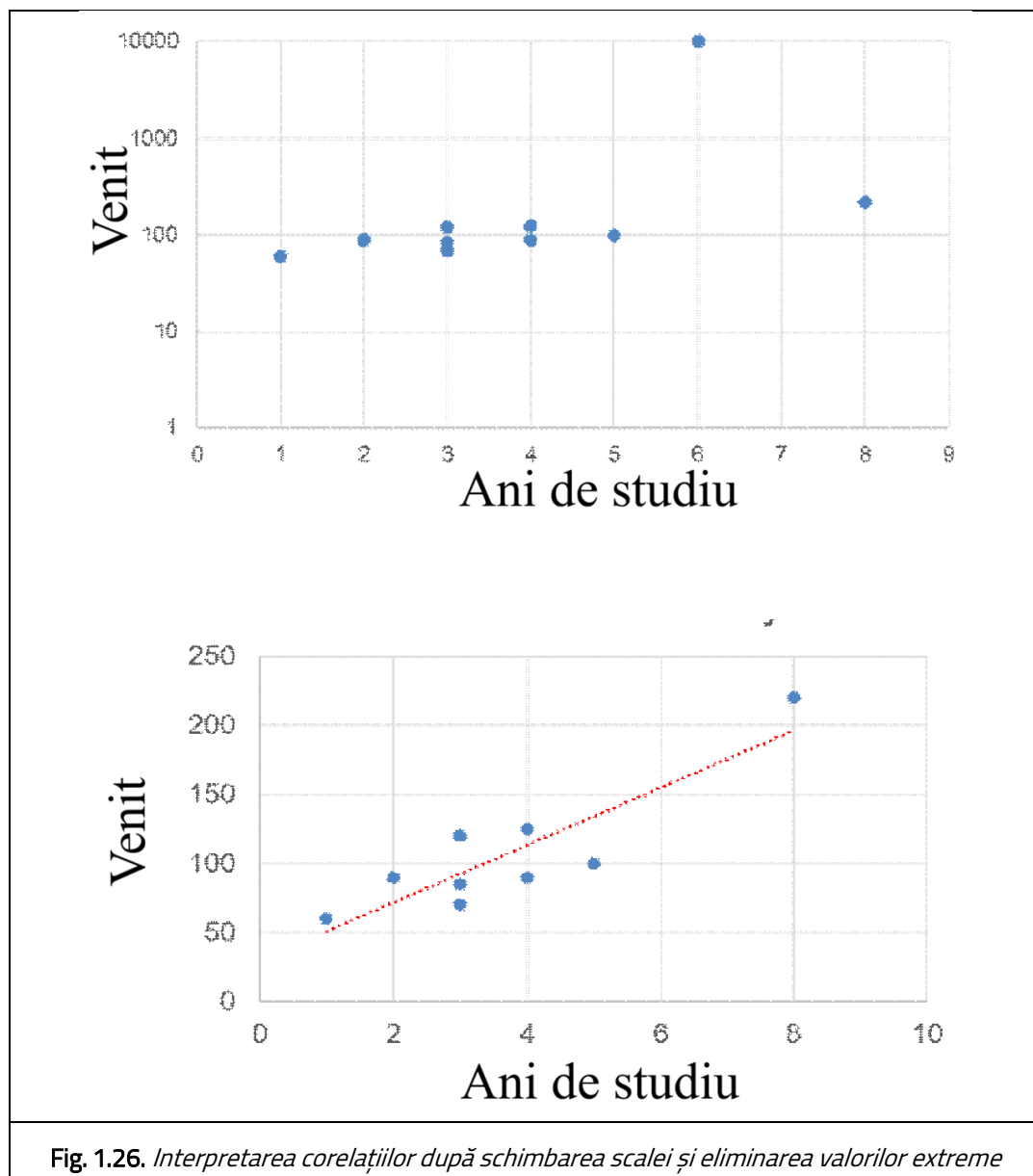


Fig. 1.26. Interpretarea corelațiilor după schimbarea scalei și eliminarea valorilor extreme

Vizualizarea este utilă și pentru atribute care evoluează în timp. În astfel de cazuri, graficele cu linie permit observarea tendințelor, creșterilor, scăderilor și diferențelor dintre serii. De exemplu, evoluția vânzărilor pentru două produse poate fi comparată ușor prin reprezentarea valorilor anuale într-un grafic comun.

Figura 1.27 prezintă evoluția vânzărilor pentru două produse pe mai mulți ani. Tabelul oferă valorile exacte, iar graficul evidențiază tendințele. Se observă creșteri succesive până la un anumit moment și o scădere accentuată la finalul intervalului. Acest exemplu arată că tabelul și graficul au roluri complementare: tabelul păstrează precizia, iar vizualizarea facilitează interpretarea.

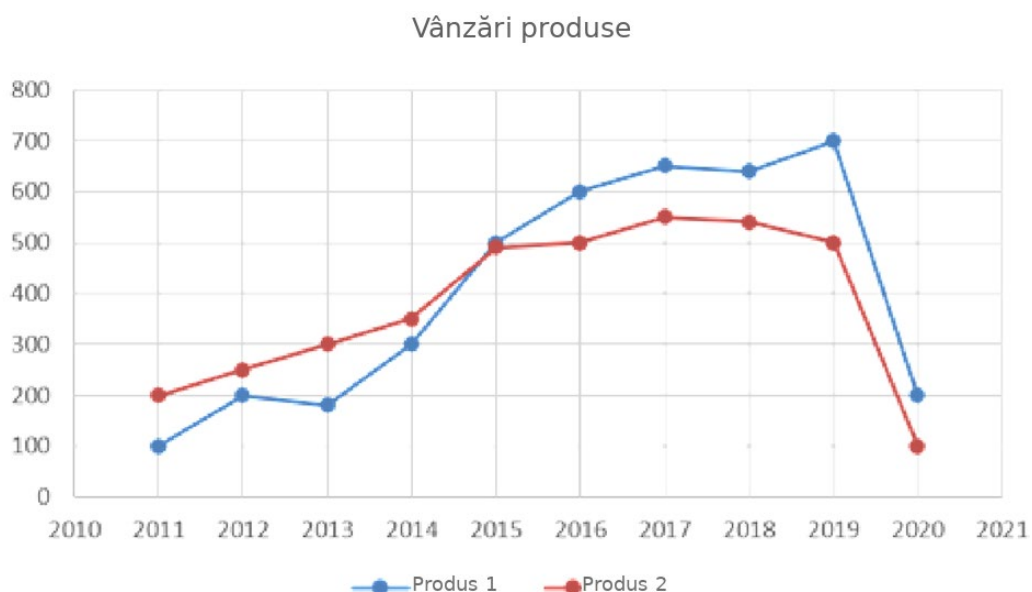


Fig. 1.27. Afișarea evoluției atributelor în timp prin grafic cu linie

2. Capitolul 2. Fundamentele învățării automate

Învățarea automată reprezintă una dintre principalele direcții ale inteligenței artificiale moderne și stă la baza multor sisteme informatice capabile să ia decizii, să recunoască tipare, să facă predicții sau să adapteze rezultatele în funcție de date. Dacă în programarea tradițională soluția este formulată explicit de către programator, în învățarea automată modelul este construit pe baza exemplurilor existente. Astfel, accentul se mută de la scrierea manuală a regulilor la descoperirea automată a relațiilor existente în date.

Acest capitol introduce conceptele de bază ale domeniului, pornind de la relația dintre inteligență artificială și învățare automată, continuând cu noțiunea de învățare din experiență, cu reprezentarea problemelor prin funcții și modele, cu principalele paradigme de învățare și cu ideea de generalizare.

2.1. Inteligența Artificială și Învățarea Automată

Inteligența artificială are ca obiectiv dezvoltarea unor sisteme capabile să realizeze sarcini care, în mod obișnuit, sunt asociate cu inteligența umană: raționament, recunoaștere, înțelegere, planificare sau decizie. În cadrul acestui domeniu larg, învățarea automată reprezintă direcția care studiază modul în care calculatoarele pot dobândi abilități utile prin experiență și prin utilizarea datelor, fără a fi programate explicit pentru fiecare situație posibilă.

Figura 2.1 ilustrează poziționarea învățării automate în contextul inteligenței artificiale. Inteligența artificială este domeniul general, reprezentarea cunoștințelor se referă la modul în care informațiile pot fi structurate și utilizate de un sistem, iar învățarea automată oferă mecanisme prin care sistemul poate extrage modele din date. În practică, aceste direcții nu sunt complet separate, ci se completează reciproc în proiectarea sistemelor inteligente.

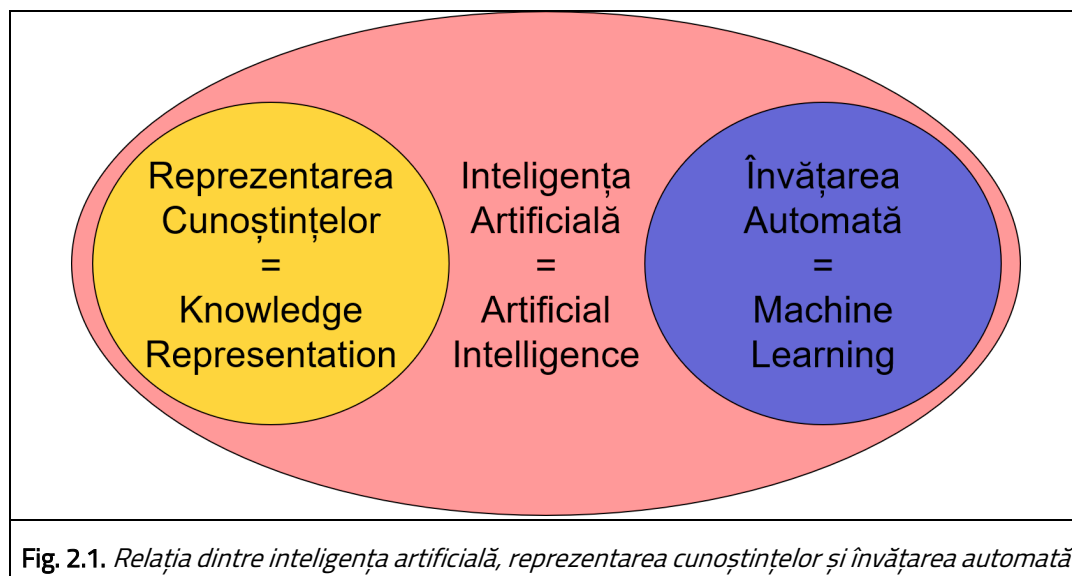


Fig. 2.1. *Relația dintre inteligența artificială, reprezentarea cunoștințelor și învățarea automată*

Un reper clasic în definirea inteligenței artificiale este Testul Turing. Conform acestui test, un sistem poate fi considerat inteligent dacă, în cadrul unei conversații, un interlocutor uman nu poate distinge în mod sigur dacă răspunsurile provin de la un om sau de la un calculator. Figura 2.2 prezintă această idee printr-o schemă intuitivă: evaluatorul comunică indirect cu doi interlocutori, iar sistemul informatic este considerat convingător dacă produce un comportament conversațional comparabil cu cel uman.

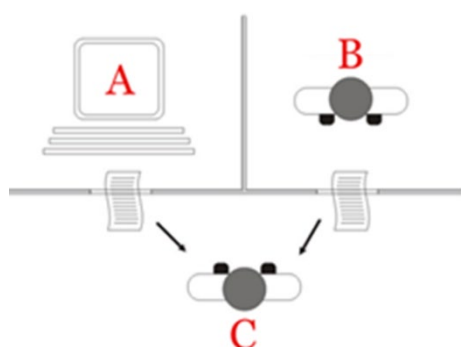


Fig. 2.2. *Inteligența artificială și ideea Testului Turing*

Învățarea automată pornește de la observația că multe sarcini sunt prea complexe pentru a fi descrise printr-un set complet de reguli. De exemplu, este dificil să se scrie manual toate regulile prin care un sistem să identifice o față într-o fotografie, să

recunoască o cifră scrisă de mână sau să decidă dacă un mesaj este spam. În schimb, pot fi colectate exemple, iar sistemul poate învăța relațiile dintre intrări și ieșiri.

Figura 2.3 prezintă diferența dintre raționamentul deductiv și cel inductiv. În raționamentul deductiv, procesul pornește de la o teorie generală, formulează ipoteze și verifică observațiile. În raționamentul inductiv, procesul pornește de la observații, identifică un model și ajunge la ipoteze sau teorii generale. Învățarea automată este strâns legată de raționamentul inductiv, deoarece algoritmi pornesc de la exemple concrete și încearcă să construiască o regulă generală.

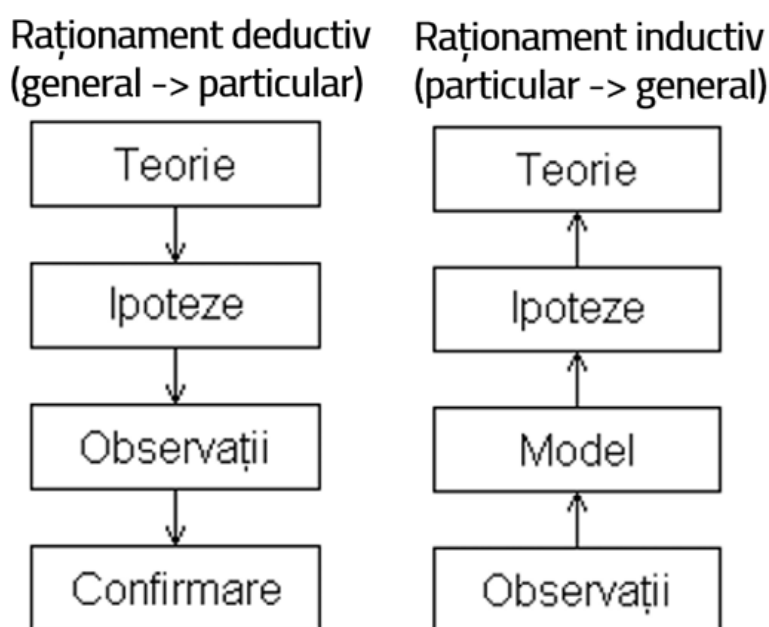


Fig. 2.3. *Raționamentul deductiv și raționamentul inductiv*

Într-o formulare intuitivă, învățarea înseamnă îmbunătățirea performanței prin experiență. Herbert Simon definea învățarea ca "orice proces prin care un sistem își îmbunătățește performanța pe baza experienței". În aceeași direcție, Tom Mitchell a formulat una dintre cele mai cunoscute definiții ale învățării automate: "un program învață din experiența E, cu privire la o clasă de sarcini T și o măsură de performanță P, dacă performanța la sarcinile din T, măsurată prin P, se îmbunătățește odată cu experiența E".

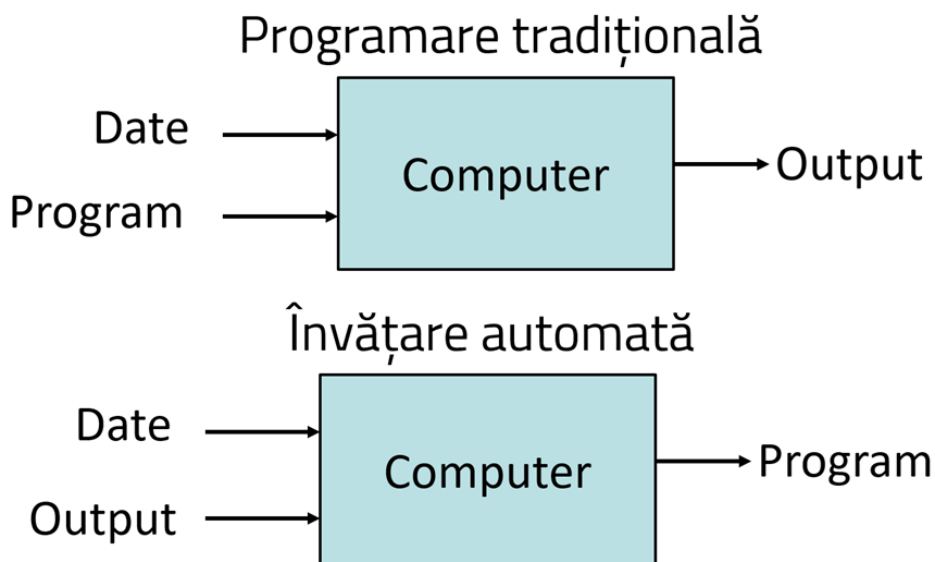


Fig. 2.4. Diferența dintre programarea tradițională și paradigma învățării automate

Figura 2.4 compară programarea tradițională cu paradigma învățării automate. În programarea tradițională, calculatorul primește datele și programul, iar rezultatul este ieșirea determinată. În învățarea automată, computerul primește date și exemple de ieșire, iar rezultatul procesului de antrenare este un model sau un program capabil să producă ieșiri pentru date noi. Această inversare a perspectivei explică de ce învățarea automată este utilă în probleme în care regulile sunt greu de formulat explicit.

2.2. Date și funcții în învățarea automată

În centrul învățării automate se află ideea de a învăța o funcție. Se consideră un set de intrări X și un set de ieșiri Y , iar scopul algoritmului este să găsească o regulă care asociază fiecărei intrări o ieșire potrivită. În forma cea mai simplă, această relație poate fi scrisă ca $f: X \rightarrow Y$. Funcția f reprezintă relația reală, de obicei necunoscută, iar modelul construit de algoritm este o aproximare a acesteia.

De exemplu, într-o problemă de filtrare a mesajelor spam, intrarea X este un email, iar ieșirea Y poate fi una dintre cele două clase: spam sau non-spam. Într-o problemă de recunoaștere a cifrelor, intrarea este imaginea unei cifre scrise de mână, iar ieșirea este una dintre clasele 0, 1, 2, ..., 9. De asemenea, într-o problemă de regresie, intrarea poate

fi o descriere numerică a unei situații, iar ieșirea este o valoare continuă care trebuie prezisă.

Figura 2.5 explică de ce învățarea automată a devenit atât de importantă în prezent. Creșterea puterii de calcul, disponibilitatea unor volume mari de date și dezvoltarea unor modele mai bune au făcut posibilă obținerea unor performanțe ridicate. Graficul sugerează faptul că, pe măsură ce numărul exemplelor de antrenare crește, acuratețea modelului se poate îmbunătăți până la un anumit nivel, după care progresul devine mai lent.

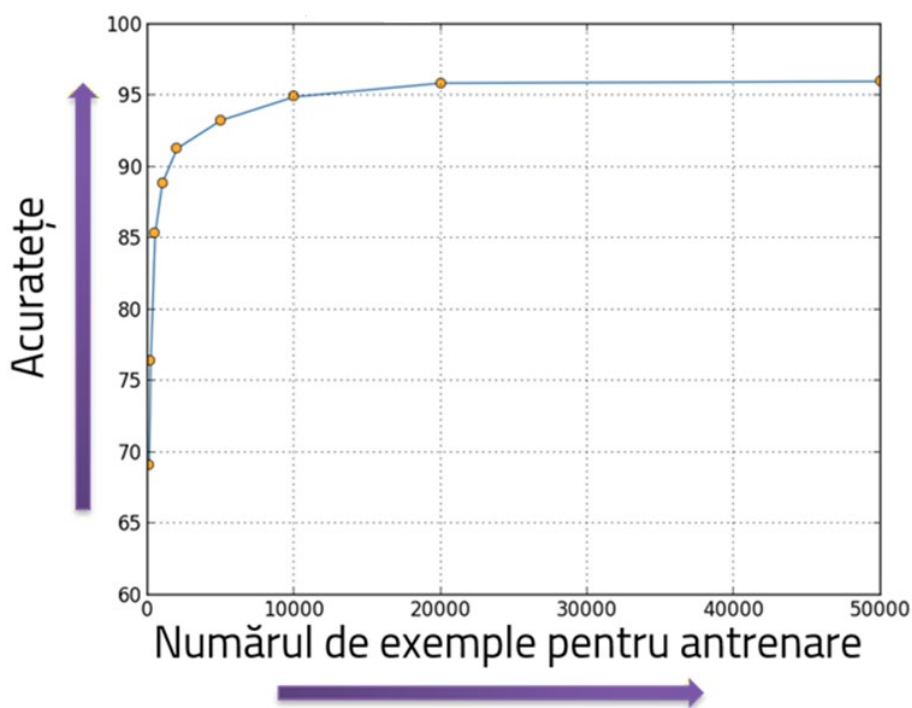


Fig. 2.5. Influența datelor, puterii de calcul și modelelor asupra performanței sistemelor ML

Așadar, învățarea automată studiază algoritmi care se pot îmbunătăți automat prin experiență și prin utilizarea datelor. Această îmbunătățire poate însemna învățarea unor parametri, a unor structuri sau a unor concepte ascunse. În funcție de problemă, modelul poate învăța probabilități, reguli, limite de decizie, reprezentări interne sau grupări ale datelor.

Un model de învățare automată nu trebuie confundat cu datele în sine. Datele reprezintă exemplele pe baza cărora se realizează antrenarea. Modelul reprezintă rezultatul procesului de învățare și poate fi aplicat ulterior asupra unor date noi. Scopul nu este memorarea exemplelor, ci identificarea unei reguli care funcționează și pentru situații care nu au fost observate anterior.

Intrarea X este de obicei un vector de caracteristici, numite și atribute, variabile independente sau regresori. Ieșirea Y este variabila țintă, adică rezultatul care trebuie prezis. În regresie, Y este o variabilă cantitativă, precum o valoare numerică sau un preț. În clasificare, Y aparține unei mulțimi finite de clase, cum ar fi spam/non-spam, pozitiv/negativ sau una dintre mai multe categorii posibile.

Datele de antrenare sunt formate din perechi de forma $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, unde fiecare x_i reprezintă o instanță, iar y_i reprezintă ieșirea corectă asociată. Pe baza acestor exemple, algoritmul caută un model g care să aproximeze cât mai bine funcția reală f . Când modelul primește o instanță nouă x , el produce o predicție $y = g(x)$.

Relația dintre învățarea automată și Data Mining este foarte strânsă. Ambele folosesc adesea metode similare, însă scopurile pot fi diferite. Învățarea automată este orientată în special spre predicție și generalizare, în timp ce Data Mining accentuează descoperirea de proprietăți, tipare sau relații necunoscute anterior în colecții mari de date. În practică, cele două domenii sunt înrudite și utilizate în fluxuri comune de analiză a datelor.

2.3. Învățarea supervizată: clasificare și regresie

Învățarea supervizată este paradigma în care algoritmul primește exemple etichetate. Fiecare exemplu conține o intrare și ieșirea corectă asociată. Scopul este de a deduce o funcție care poate prezice ieșirea pentru instanțe noi, nevăzute în timpul antrenării. Calitatea modelului este evaluată prin capacitatea lui de generalizare, adică prin performanța obținută pe date care nu au fost folosite pentru construirea modelului.

Figura 2.6 prezintă un exemplu de clasificare a imaginilor. Sistemul primește imagini etichetate cu clase precum mașină, persoană sau câine. Pe baza acestor exemple, modelul trebuie să învețe caracteristici relevante, astfel încât să poată atribui corect o etichetă unei imagini noi. Aceasta este o problemă tipică de clasificare, deoarece ieșirea aparține unei mulțimi finite de clase.



Fig. 2.6. Exemplu de clasificare a imaginilor pe baza etichetelor disponibile

Figura 2.7 ilustrează filtrarea spam-urilor. Intrarea este un email, iar ieșirea este clasa spam sau ham, adică mesaj legitim. Pentru a construi un asemenea sistem, este necesară o colecție de mesaje etichetate manual. Din mesaje sunt extrase caracteristici, precum apariția unor cuvinte, modele de scriere, prezența literelor mari sau informații despre expeditor. Modelul utilizează aceste caracteristici pentru a decide clasa mesajelor viitoare.

Input: email

Output: spam/ham

Setup:

- Get a large collection of example emails, each labeled "spam" or "ham"
- Note: someone has to hand label all this data!
- Want to learn to predict labels of new, future emails

Features: The attributes used to make the ham / spam decision

- Words: FREE!
- Text Patterns: \$dd, CAPS
- Non-text: SenderInContacts
- ...



Dear Sir.

First, I must solicit your confidence in this transaction, this is by virtue of its nature as being utterly confidential and top secret. ...



TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES FOR ONLY \$99



Ok, I know this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Fig. 2.7. Filtrarea mesajelor spam ca problemă de clasificare binară

Figura 2.8 prezintă recunoașterea cifrelor scrise de mână. În cazul setului de date MNIST, fiecare imagine are 28×28 pixeli, deci poate fi reprezentată printr-un vector cu 784 de componente. Modelul trebuie să transforme acest vector într-o etichetă numerică, de la 0 la 9. Problema este foarte utilă din punct de vedere didactic, deoarece arată cum o imagine poate fi convertită într-o reprezentare numerică prelucrabilă de algoritmi.

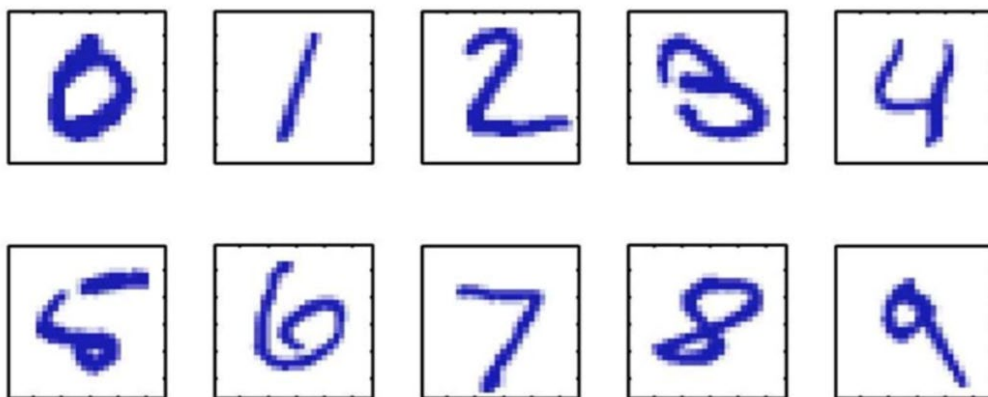


Fig. 2.8. *Recunoașterea cifrelor scrise de mână pe baza imaginilor MNIST*

Figura 2.9 prezintă o abordare intuitivă pentru recunoașterea facială. O fereastră este deplasată peste imagine, iar fiecare regiune este clasificată ca față sau alt element. Problema inițială de detecție este transformată astfel într-o problemă de clasificare binară. Ideea este importantă deoarece multe sarcini complexe pot fi transformate în probleme mai simple, pentru care există algoritmi consacrați.

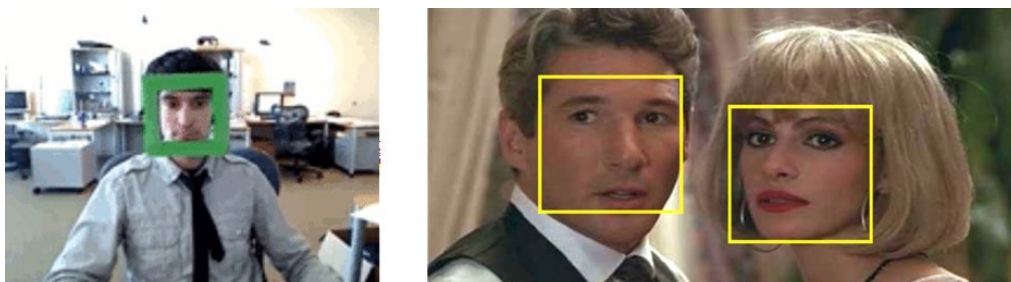


Fig. 2.9. *Recunoașterea facială prin clasificarea unor ferestre din imagine.*

Figura 2.10 prezintă un exemplu de predicție a unei valori numerice, cum ar fi estimarea unui preț la o dată viitoare. Aceasta este o sarcină de regresie, deoarece ieșirea nu este o clasă discretă, ci o valoare continuă. Regresia este utilizată atunci când dorim să aproximăm o relație numerică între variabile și să estimăm valori necunoscute pe baza observațiilor existente.

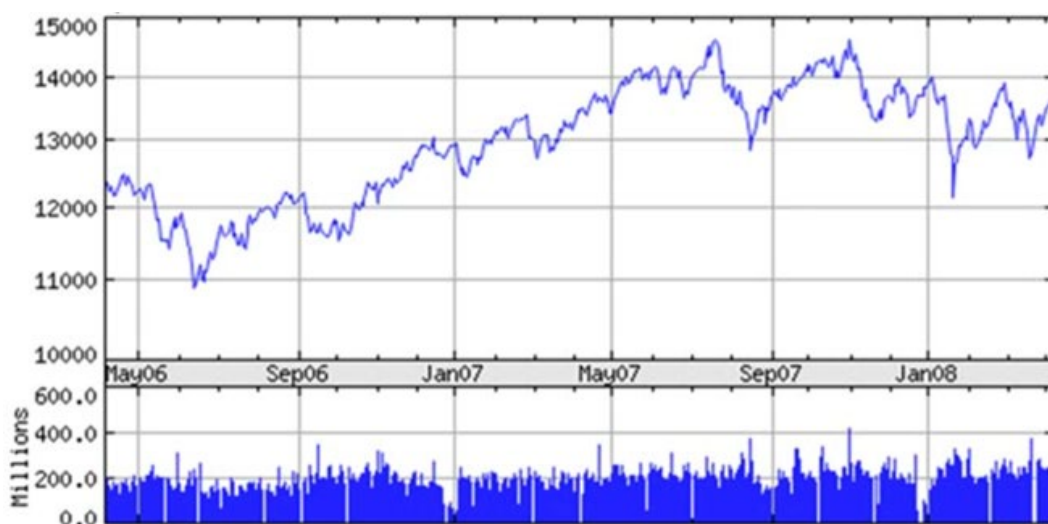


Fig. 2.10. *Predictia unei valori continue ca problemă de regresie*

Figura 2.11 compară cele două forme canonice ale învățării supervizate. În clasificare, modelul separă exemplele în clase distincte, reprezentate vizual prin grupuri de puncte diferite. În regresie, modelul caută o funcție care aproximează relația dintre variabile, de exemplu o dreaptă care descrie tendința generală a punctelor. Diferența esențială este natura ieșirii: clasă discretă în clasificare și valoare continuă în regresie.

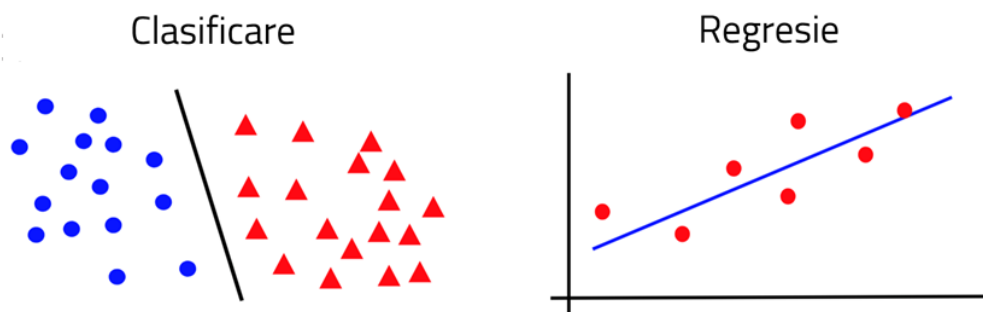


Fig. 2.11. *Forme canonice pentru clasificare și regresie*

Figura 2.12 sintetizează paradigma de învățare supervizată.

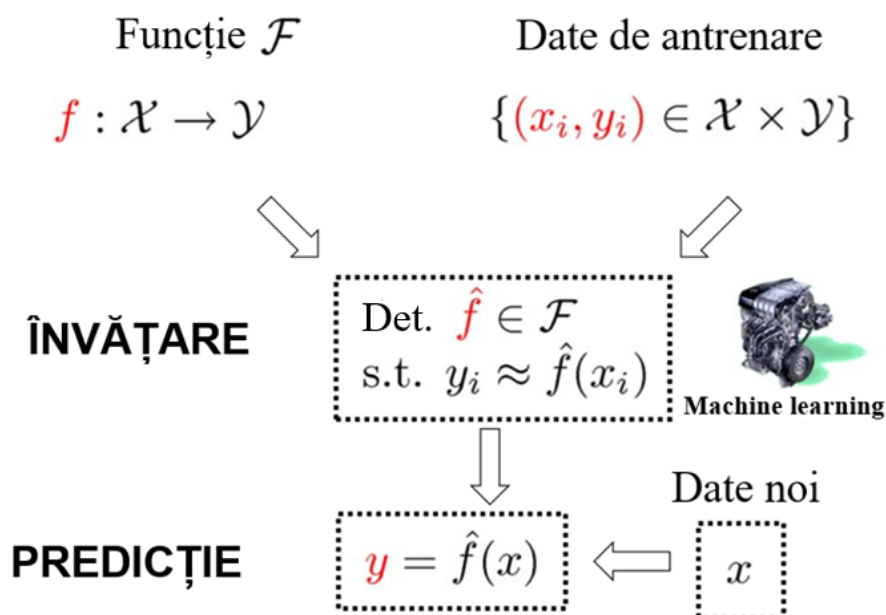


Fig. 2.12. Paradigma de învățare supervizată: antrenare și predicție

În etapa de învățare se pornește de la date de antrenare formate din perechi intrare-ieșire și se caută o funcție aproximativă. În etapa de predicție, modelul este aplicat asupra unei intrări noi și produce o ieșire estimată. Această schemă este comună pentru numeroși algoritmi, de la metode simple, precum KNN, până la rețele neuronale complexe.

Printre algoritmi utilizați în învățarea supervizată se numără clasificatorul Bayes naiv, metoda celor mai apropiați vecini, mașinile cu vectori suport, arborii de decizie, *Random Forest* și rețelele neuronale. Alegerea algoritmului depinde de tipul problemei, dimensiunea datelor, natura caracteristicilor, nivelul de interpretabilitate dorit și performanța obținută la testare.

2.4. Învățarea nesupervizată și alte paradigme de învățare

Învățarea nesupervizată se aplică atunci când datele nu au etichete asociate. Algoritmul nu primește ieșiri corecte, ci trebuie să descopere singur structuri, grupări sau reprezentări utile. Această paradigmă este folosită frecvent pentru explorarea datelor, pentru clustering, pentru reducerea dimensionalității și pentru identificarea unor concepte latente.

Figura 2.13 prezintă ideea de bază a învățării nesupervizate: sunt disponibile obiecte fără etichete, iar sistemul trebuie să le organizeze în funcție de asemănări. De exemplu, imaginile pot fi grupate în funcție de conținut vizual, chiar dacă nu se oferă explicit clase precum mașină, persoană sau animal. Rezultatul depinde de reprezentarea datelor și de măsura de similaritate folosită.



Fig. 2.13. *Învățarea nesupervizată prin gruparea imaginilor după similaritate*

Figura 2.14 exemplifică aplicarea clustering-ului asupra imaginilor MNIST. Punctele colorate reprezintă imagini proiectate într-un spațiu bidimensional, iar grupările sugerează faptul că imagini similare tind să se afle aproape unele de altele. Chiar dacă etichetele nu sunt utilizate pentru antrenare, structura datelor poate reflecta clasele reale.

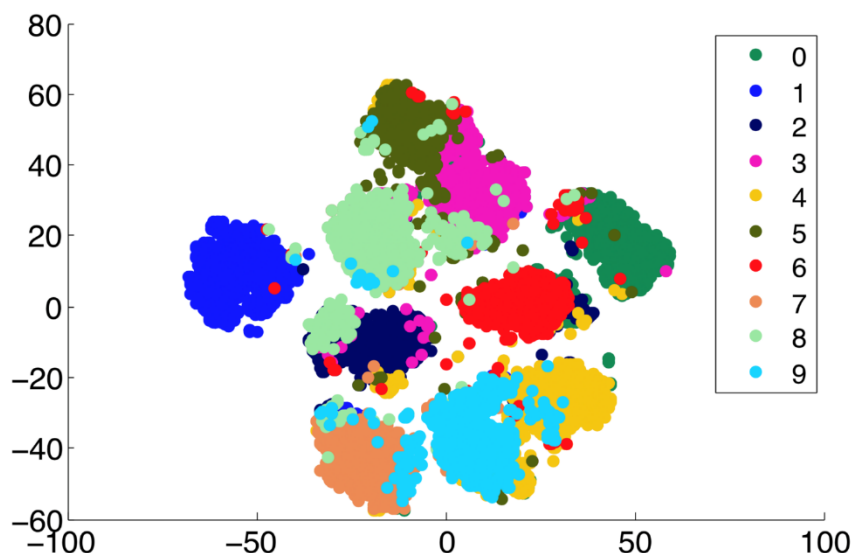


Fig. 2.14. *Clustering aglomerativ aplicat asupra imaginilor MNIST*

Figura 2.15 prezintă principiul unui autoencoder. Intrarea este comprimată de un codificator într-o reprezentare mai mică, numită uneori *bottleneck*, iar un decodificator încearcă să reconstruiască ieșirea. În exemplu, imaginea zgomotoasă este transformată într-o reprezentare compactă și apoi reconstruită într-o formă mai clară. Autoencoderele sunt utile pentru reducerea dimensionalității, eliminarea zgomotului și învățarea de caracteristici.

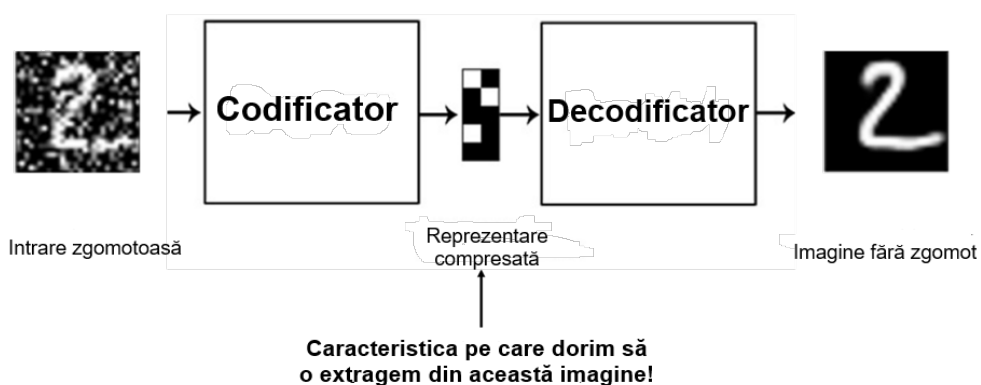


Fig. 2.15. *Autoencoder pentru învățarea de reprezentări prin principiul bottleneck*

Figura 2.16 prezintă clustering-ul ierarhic printr-o dendrogramă. Obiectele sunt organizate pe niveluri, în funcție de similaritate. La niveluri inferioare se observă grupuri apropiate, iar la niveluri superioare se obțin grupări mai generale. O astfel de reprezentare este utilă atunci când relațiile dintre obiecte sunt graduale sau ierarhice.

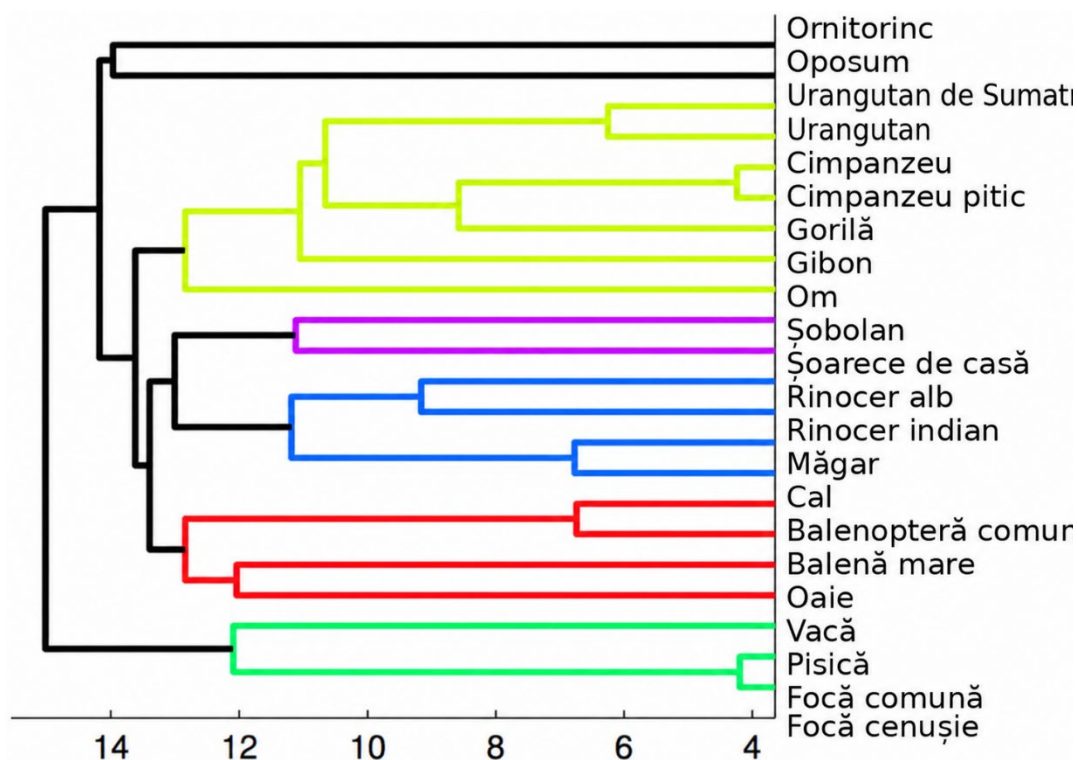


Fig. 2.16. Clustering ierarhic reprezentat printr-o dendrogramă

Figura 2.17 sintetizează două forme canonice ale învățării nesupervizate. În clustering, punctele sunt împărțite în grupuri în funcție de similaritate. În reducerea dimensionalității, datele sunt proiectate într-un spațiu cu mai puține dimensiuni, păstrând cât mai mult din structura lor. Această reducere permite vizualizarea, compresia și uneori îmbunătățirea performanței modelelor ulterioare.

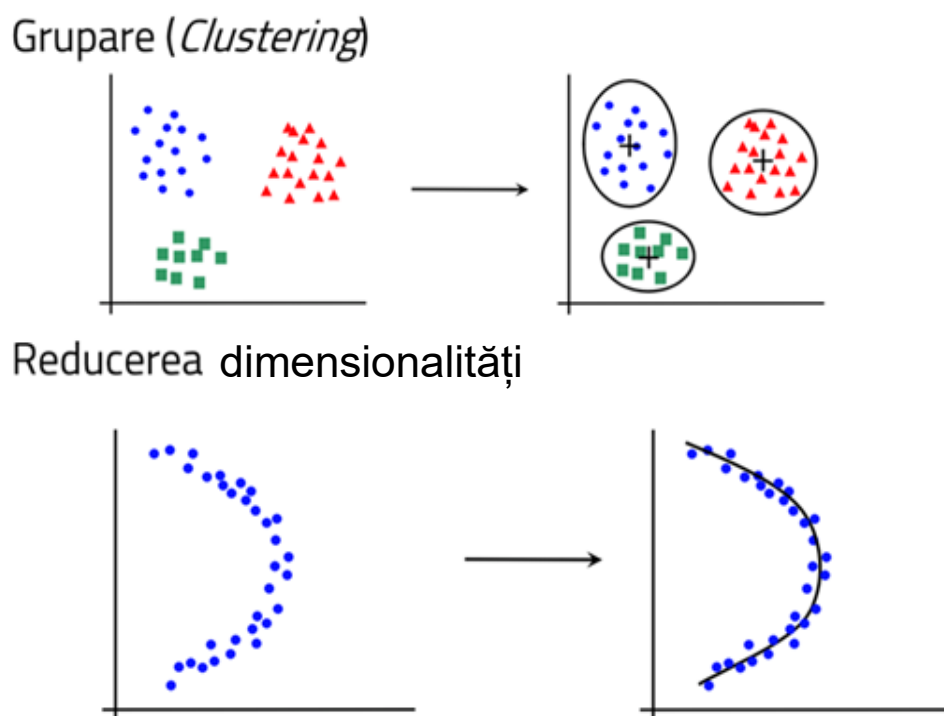


Fig. 2.17. *Forme canonice ale învățării nesupervizate: clustering și reducerea dimensionalității*

Figura 2.18 prezintă învățarea semi-supervizată. În acest caz, sistemul dispune de un număr redus de exemple etichetate și de un număr mai mare de exemple neetichetate.



Fig. 2.18. *Învățarea semi-supervizată cu exemple etichetate și neetichetate*

Această situație apare frecvent în practică, deoarece etichetarea manuală a datelor poate fi costisitoare. Algoritmul încearcă să valorifice atât informația explicită din exemplele etichetate, cât și structura prezentă în datele netichetate.

O altă paradigmă importantă este învățarea prin consolidare, numită și *reinforcement learning*. În acest caz, un agent interacționează cu un mediu, observă starea acestuia, execută acțiuni și primește recompense sau penalizări. Spre deosebire de învățarea supervizată, răspunsul corect nu este oferit imediat pentru fiecare exemplu, iar agentul trebuie să învețe o strategie de acțiune în timp.

Figura 2.19 ilustrează un exemplu celebru de învățare prin consolidare: învățarea jocului Go. Agentul primește recompensă pozitivă sau negativă în funcție de câștigarea sau pierderea jocului. Complexitatea este dată de faptul că o mutare poate avea efecte pe termen lung, iar recompensa finală este obținută după o secvență lungă de acțiuni.

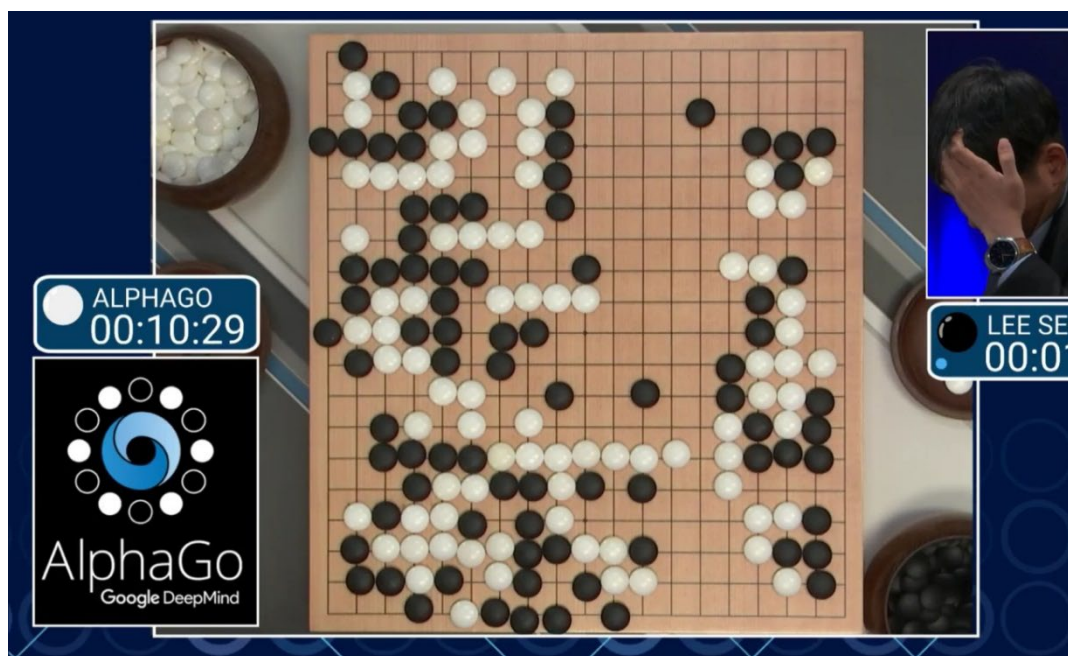


Fig. 2.19. Exemplu de învățare prin consolidare: jocul „Go”

Figura 2.20 prezintă un alt exemplu de reinforcement learning: un robot care învață să meargă pe bicicletă. Recompensa poate fi asociată cu deplasarea înainte, iar penalizarea cu pierderea echilibrului sau căderea. Acest exemplu arată că învățarea prin consolidare este relevantă pentru control, robotică și sisteme autonome.

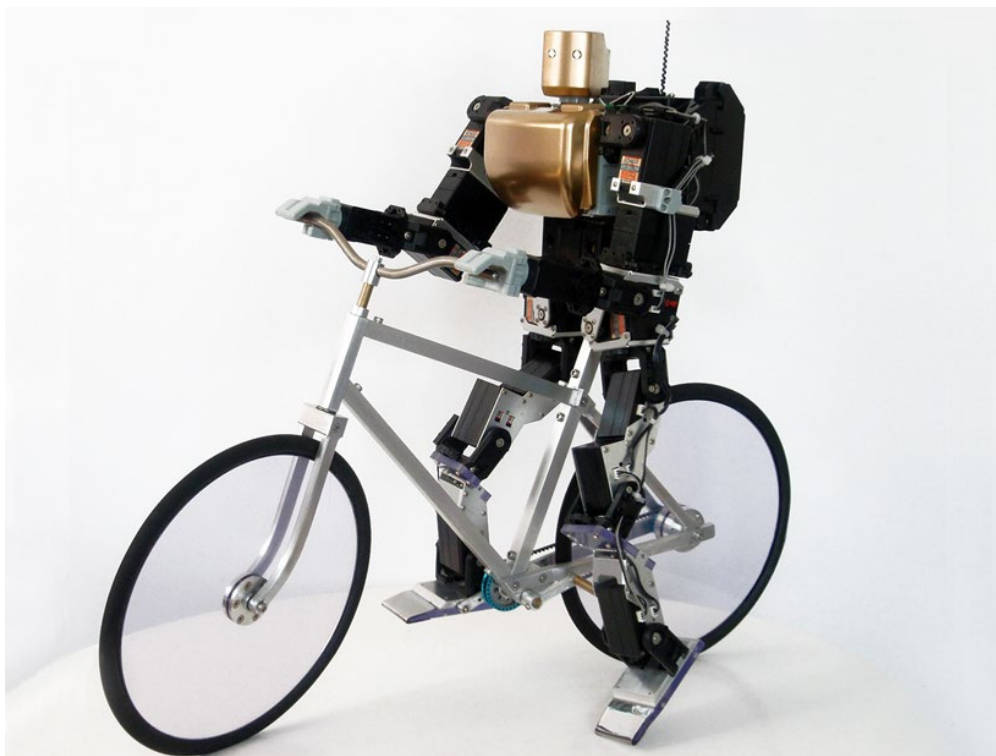


Fig. 2.20. *Exemplu de învățare prin consolidare: robot care învață să meargă pe bicicletă*

Figura 2.21 prezintă învățarea jocului Pong pornind de la informații vizuale. Agentul observă pixelii jocului, alege acțiuni și primește recompense în funcție de scor. Acest tip de abordare este important deoarece leagă percepția de decizie și arată cum datele brute pot fi folosite pentru învățarea unui comportament.

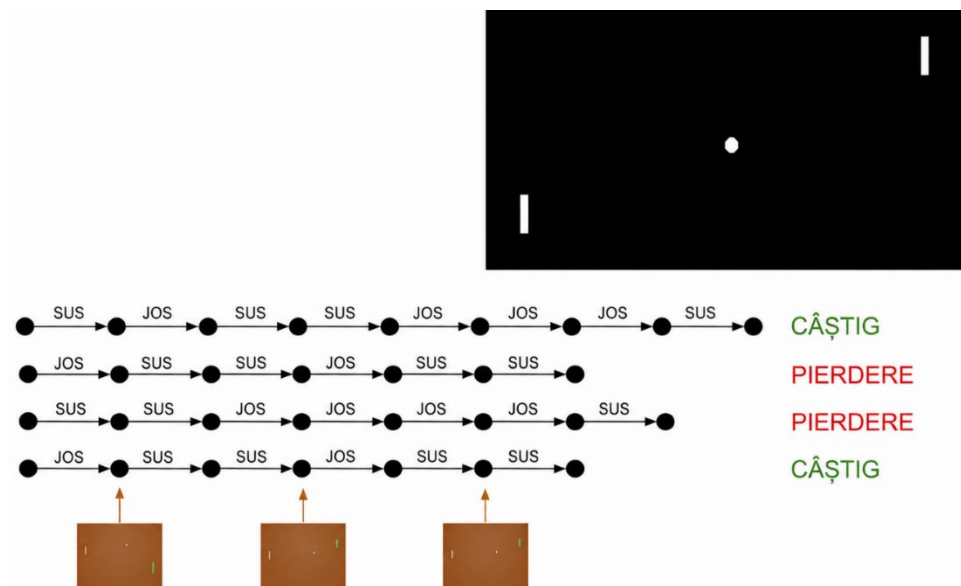


Fig. 2.21. Exemplu de învățare prin consolidare: jocul Pong învățat din pixeli

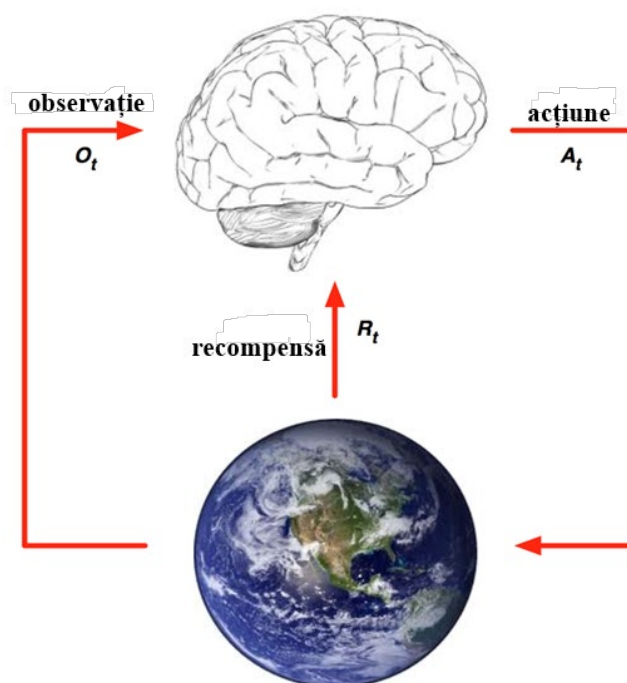


Fig. 2.22. Paradigma reinforcement learning

Figura 2.22 sintetizează paradigma reinforcement learning. Agentul observă mediul, execută o acțiune, iar mediul returnează o recompensă și o nouă observație. Procesul este secvențial, iar deciziile curente influențează datele viitoare. Din acest motiv, timpul și succesiunea acțiunilor au un rol esențial.

Figura 2.23 prezintă învățarea activă. Atunci când există un set mare de exemple netichetate, sistemul poate selecta un subset redus de exemple pentru etichetare. Scopul este obținerea unui model cât mai bun cu un cost cât mai mic de etichetare. În această paradigmă, modelul participă activ la alegerea datelor care vor fi adnotate.

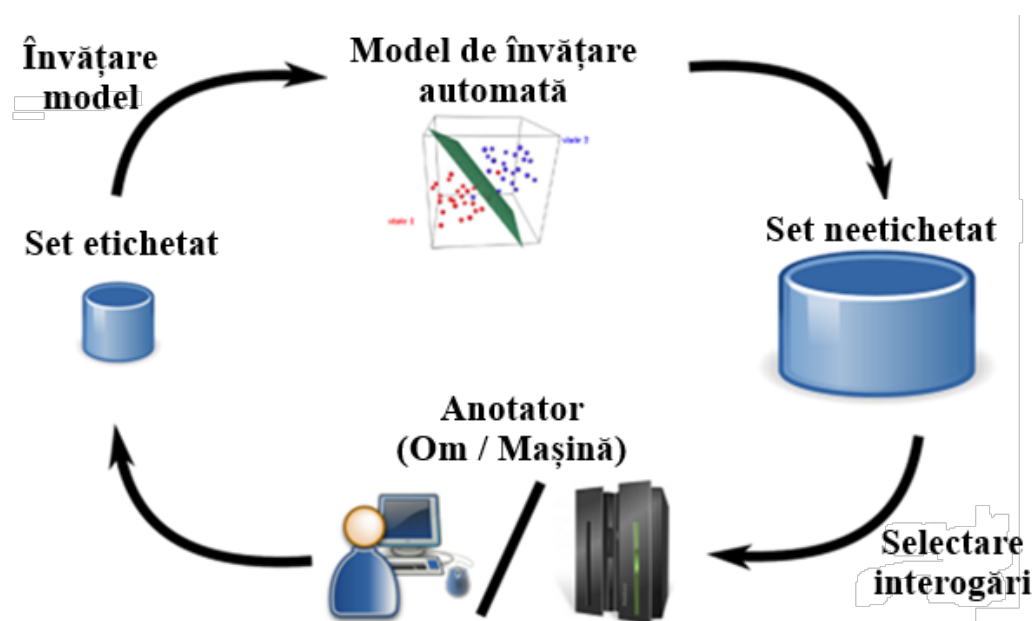


Fig. 2.23 Învățarea activă prin selectarea exemplurilor care trebuie etichetate

Figura 2.24 prezintă paradigma învățării prin transfer. Un model antrenat pe un domeniu sau pe o problemă este reutilizat pentru o problemă apropiată. De exemplu, o rețea neuronală convoluțională antrenată pentru recunoașterea unor clase generale de obiecte poate fi adaptată ulterior pentru clase mai specifice. În acest caz, modelul păstrează cunoștințele deja învățate, cum ar fi detectarea marginilor, formelor, texturilor sau a unor tipare vizuale generale, iar apoi își ajustează parametrii pentru noua sarcină. Această abordare este foarte utilă atunci când datele disponibile pentru noua problemă sunt limitate.

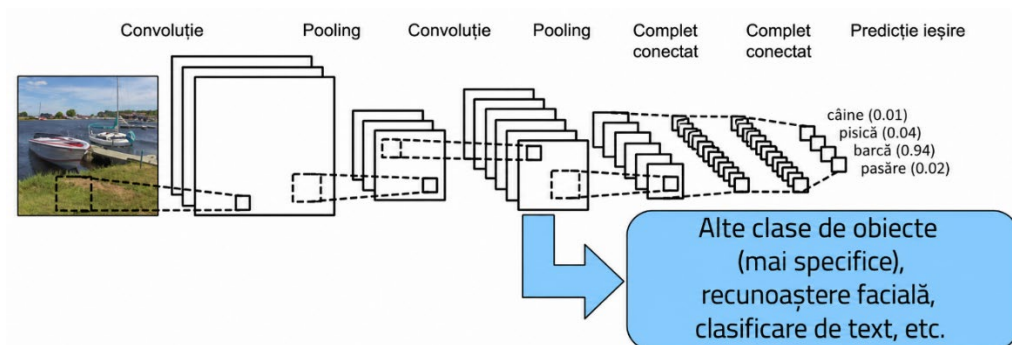


Fig. 2.24. *Învățarea prin transfer pornind de la un model antrenat anterior*

2.5. Generalizare

În învățarea automată, noțiunea de generalizare reprezintă capacitatea unui model de a produce predicții corecte pentru date noi, diferite de exemplele pe care le-a văzut în etapa de antrenare. Acesta este practic scopul real al unui model de Machine Learning: nu memorarea datelor de antrenament, ci identificarea unor tipare suficient de stabile pentru a fi utilizate în situații viitoare. Un model care are o capacitate bună de generalizare este un model care a extras informația relevantă din date și o poate aplica asupra unor instanțe noi [21].

Această idee este esențială deoarece datele disponibile pentru antrenare reprezintă, de regulă, doar un eșantion limitat din realitatea pe care dorim să o modelăm. În practică, un sistem de clasificare, regresie sau recomandare va fi utilizat pe date viitoare, provenite de la utilizatori, senzori, documente sau procese care nu au fost incluse în setul inițial. Prin urmare, performanța obținută exclusiv pe setul de antrenare nu este suficientă pentru a afirma că modelul este bun.

Figura 2.25 ilustrează ideea de generalizare. Imaginea prezintă mai multe exemple diferite de câini, aparținând unor rase variate, cu forme, dimensiuni, culori și poziții diferite. Deși aceste exemple nu sunt identice, ele au caracteristici comune care permit unui model să învețe conceptul general de „câine”.

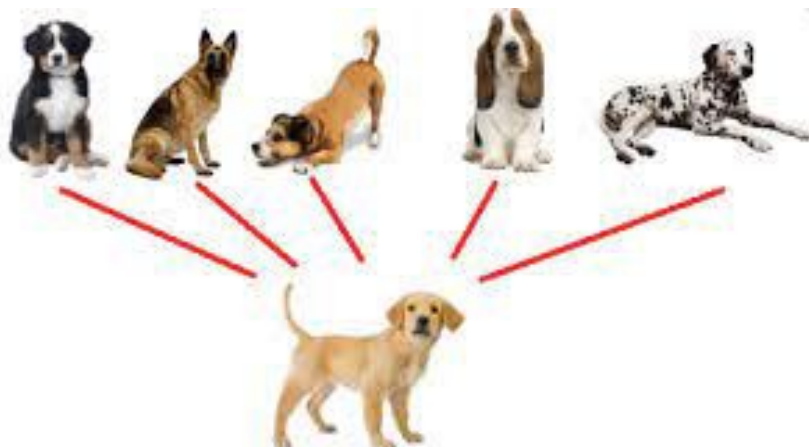


Fig. 2.25. *Generalizarea și ideea unei ipoteze simple care funcționează pe exemple noi*

Exemplele de antrenare pot fi memorate, însă memorarea nu garantează capacitatea de a clasifica instanțe noi. Briciul lui Occam sugerează că, atunci când există mai multe explicații posibile, este preferabilă o ipoteză mai simplă, dacă aceasta este suficient de consistentă cu datele. În învățarea automată, modelele prea complexe pot deveni sensibile la particularități irelevante ale datelor.

Pentru a înțelege generalizarea, este utilă distincția dintre eroarea de antrenare și eroarea de generalizare. Eroarea de antrenare este calculată pe exemplele utilizate pentru construirea modelului. Eroarea de generalizare, în schimb, exprimă performanța așteptată pe date noi, provenite din aceeași problemă, dar care nu au influențat procesul de învățare.

Capacitatea de generalizare este influențată în primul rând de calitatea și reprezentativitatea datelor. Dacă datele de antrenare nu reflectă situațiile reale în care modelul va fi utilizat, performanța pe date noi poate fi slabă chiar dacă algoritmul este corect implementat. Pentru o generalizare bună, exemplele din seturile de antrenare, validare și testare trebuie să provină, pe cât posibil, din aceeași distribuție de date și să fie independente între ele. Un al doilea factor este complexitatea modelului. Modelele foarte simple au bias mare: ele impun ipoteze rigide și pot rata structuri importante. Modelele foarte complexe au varianță mare: ele pot răspunde puternic la mici variații ale setului de antrenare. Compromisul bias-varianță explică de ce nu este întotdeauna avantajos să se aleagă cel mai complex algoritm disponibil. Un model mai simplu, dar

bine validat, poate fi mai util decât unul sofisticat care se potrivește perfect pe datele antrenare și eșuează în utilizarea practică.

Generalizarea poate fi îmbunătățită prin mai multe strategii. Prima strategie este creșterea calității datelor: eliminarea erorilor evidente, tratarea valorilor lipsă, reducerea duplicatelor, includerea unor exemple reprezentative și evitarea dezechilibrelor majore între clase. A doua strategie este alegerea unei complexități adecvate a modelului. În loc să se urmărească doar minimizarea erorii de antrenare, trebuie urmărită minimizarea erorii pe date noi. Acesta este motivul pentru care, în practică, evaluarea pe seturi de validare și testare este la fel de importantă ca antrenarea propriu-zisă.

3. Capitolul 3. Tehnici de învățare supervizată: Clasificare

Capitolul prezintă principalele concepte și tehnici asociate problemelor de clasificare din învățarea supervizată. Sunt discutate diferențele dintre clasificare și regresie, modul de construire a unui model de clasificare, rolul datelor de antrenare, tipurile de attribute utilizate în procesul de decizie și condițiile necesare pentru o învățare eficientă. În continuare sunt analizate trei direcții importante: arborii de decizie, clasificarea bazată pe instanțe prin algoritmul K-Nearest Neighbors și metoda Support Vector Machine. Partea finală introduce elemente de evaluare a modelelor de clasificare, precum acuratețea, precizia, recall și scorul F1.

3.1. Clasificarea ca problemă de învățare supervizată

În învățarea supervizată, algoritmul primește exemple pentru care se cunoaște deja rezultatul corect. Fiecare exemplu este format dintr-un set de caracteristici, numite și attribute, și o valoare țintă. În cazul clasificării, valoarea țintă este o clasă discretă. Astfel, algoritmul învață din exemple etichetate și construiește un model care poate atribui clase unor instanțe noi, pentru care eticheta nu este cunoscută.

Clasificarea trebuie diferențiată de regresie. În clasificare se urmărește învățarea unei funcții discrete, adică o funcție care returnează o etichetă dintr-o mulțime finită de clase. În regresie, rezultatul este o valoare continuă, de exemplu o temperatură, o distanță sau o valoare numerică estimată. O problemă de clasificare binară are doar două clase posibile, cum ar fi pozitiv/negativ, adevărat/fals sau da/nu. O problemă multi-clasă poate avea trei sau mai multe clase.

Un exemplu intuitiv este decizia privind desfășurarea unui joc în funcție de starea vremii. Instanțele sunt observații meteorologice, iar attributele pot fi starea vremii, temperatura, umiditatea și vântul. Ultima coloană reprezintă clasa, adică decizia dacă jocul se desfășoară sau nu. Modelul trebuie să învețe relația dintre condițiile meteo și decizia finală.

Figura 3.1 prezintă un set de date simplu pentru clasificare. Fiecare rând reprezintă o instanță, iar fiecare coloană reprezintă un atribut. Variabila țintă este clasa, în exemplu atributul „Joc”, care indică rezultatul ce trebuie prezis. Această organizare tabelară este foarte frecventă în învățarea supervizată, deoarece permite separarea clară între datele de intrare și ieșirea dorită.

Construirea unui model de clasificare presupune existența unui set de clase predeterminate. Se consideră că fiecare eșantion aparține uneia dintre aceste clase, iar apartenența este cunoscută în etapa de antrenare. Modelul rezultat poate fi reprezentat sub diferite forme: reguli de clasificare, arbori de decizie, funcții matematice sau limite geometrice de separare. După antrenare, modelul este aplicat asupra unor instanțe noi, iar rezultatele sunt evaluate prin metrici de performanță.

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

Fig. 3.1. Set de date pentru o problemă de clasificare

Procesul poate fi înțeles ca o succesiune de două etape. În prima etapă, datele etichetate sunt utilizate pentru învățare, iar algoritmul construiește un model. În a doua etapă, modelul este aplicat asupra datelor de testare sau asupra unor date complet noi. Această distincție este importantă deoarece performanța reală a modelului trebuie măsurată pe instanțe care nu au fost utilizate la antrenare.

Figura 3.2 ilustrează fluxul de bază al clasificării. Setul de date de antrenare conține exemple etichetate, iar algoritmul de învățare produce un model. Ulterior, modelul este aplicat asupra unui set de testare, în care clasa este necunoscută sau este ascunsă

pentru verificare. Procesul combină inducția, prin care modelul este învățat din exemple, și deducția, prin care modelul este folosit pentru a lua decizii asupra unor cazuri noi.

Performanța unui model de clasificare depinde în mare măsură de calitatea datelor. Înainte de antrenare, datele trebuie curățate, transformate și pregătite. Curățarea reduce zgomotul și tratează valorile lipsă. Analiza relevanței urmărește eliminarea atributelor irelevante sau redundante. Transformarea datelor poate include normalizarea valorilor numerice, discretizarea atributelor continue sau codificarea atributelor categoriale.

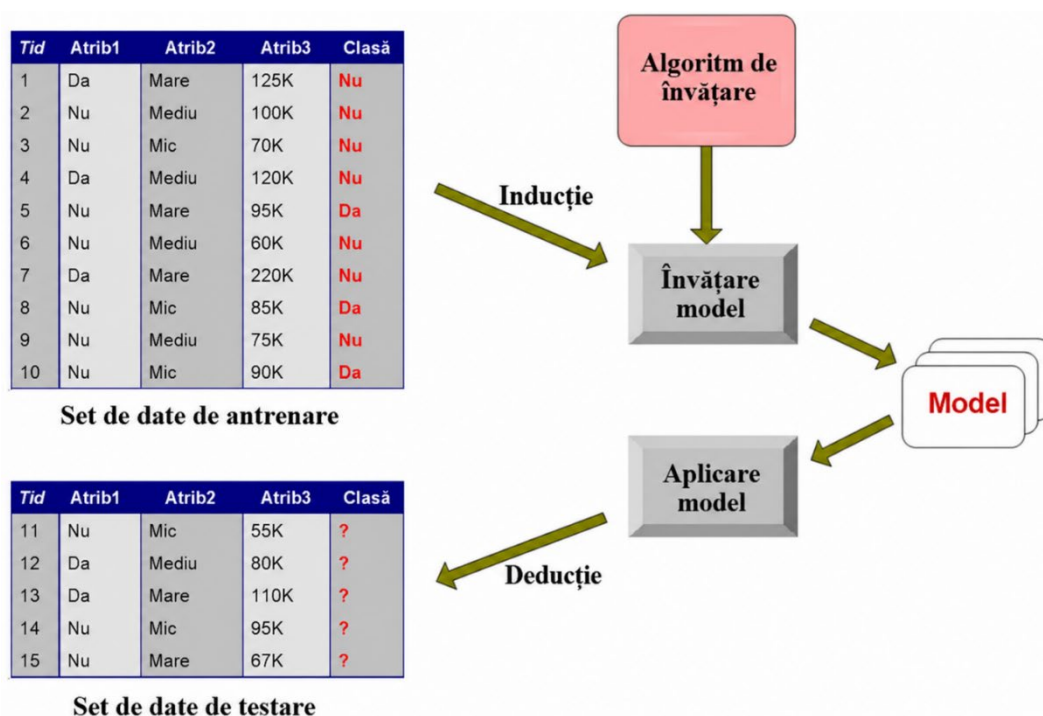


Fig. 3.2. Procesul de învățare și aplicare a unui model de clasificare

În problemele de clasificare se folosesc mai multe tipuri de atribute. Atributele nominale nu au o ordine naturală, de exemplu culoarea ochilor sau starea civilă. Atributele ordinale au o ordine, dar distanțele dintre valori nu sunt neapărat egale, cum ar fi nivelurile mic, mediu și mare. Atributele numerice pot fi de tip interval sau de tip rațional și permit operații aritmetice mai avansate. Alegerea modului de reprezentare a atributelor influențează direct algoritmul utilizat.

Pentru ca învățarea să fie bună, modelul trebuie să fie suficient de expresiv pentru a surprinde relația dintre atribute și clase, dar nu atât de complex încât să memoreze zgomotul. Dacă modelul este prea simplu, apare fenomenul de underfitting: modelul nu reușește să învețe structura datelor. Dacă modelul este prea complex, apare overfitting: modelul se potrivește foarte bine pe datele de antrenare, dar nu se generalizează bine la date noi.

Există numeroase tehnici de clasificare, fiecare având propriile avantaje, limitări și domenii de utilizare. Printre cele mai cunoscute se numără arborii de decizie, Random Forest, K-Nearest Neighbors, Support Vector Machine, regresia logistică, rețelele bayesiene și rețelele neuronale. În acest capitol sunt detaliate trei abordări fundamentale: arborii de decizie, KNN și SVM.

3.2. Arbori de decizie

Arborii de decizie sunt modele de clasificare intuitive, deoarece reprezintă procesul decizional sub forma unei succesiuni de întrebări. Fiecare nod intern al arborelui corespunde unui test asupra unui atribut, fiecare ramură corespunde unui rezultat al testului, iar fiecare frunză indică o clasă. Din acest motiv, arborii de decizie sunt ușor de explicat și de interpretat, fiind utilizați frecvent în aplicații didactice și practice.

Principiul general de construire a unui arbore de decizie este de tip greedy. Arborele este construit recursiv, de sus în jos, prin tehnica divide et impera. La început, toate instanțele se află în nodul rădăcină. Algoritmul selectează un atribut de test, împarte datele în subseturi și repetă procesul pentru fiecare subset. Scopul este obținerea unor noduri cât mai pure, adică noduri în care instanțele aparțin preponderent aceleiași clase.

Alegerea atributului rădăcină este foarte importantă. Un atribut bun separă datele astfel încât subseturile rezultate să fie mai omogene decât setul inițial. Pentru această alegere pot fi folosite măsuri euristice sau statistice, precum entropia și câștigul informațional. Entropia măsoară impuritatea unui set de date, iar câștigul informațional estimează cât de mult reduce un atribut această impuritate.

Un aspect important al arborilor de decizie este expresivitatea. În principiu, un arbore poate reprezenta funcții complexe ale atributelor de intrare. De exemplu, funcția XOR nu poate fi separată printr-o regulă liniară simplă, dar poate fi reprezentată

printr-un arbore de decizie. Totuși, expresivitatea ridicată poate deveni o problemă dacă arborele ajunge să memoreze exemplele de antrenare în loc să învețe o regulă generală.

Figura 3.3 arată că un arbore de decizie poate reprezenta funcția XOR prin testarea succesivă a atributelor A și B. Pentru fiecare combinație de valori se ajunge la o frunză corespunzătoare clasei. Exemplul este util deoarece evidențiază puterea de reprezentare a arborilor, dar și riscul de a construi arbori prea detaliați, care nu generalizează bine.

Construcția arborelui se oprește în mod obișnuit în trei situații: toate exemplele dintr-un nod aparțin aceleiași clase, nu mai există atribute disponibile pentru partiționare sau nu mai există suficiente eșantioane pentru a continua împărțirea. În practică, se pot introduce criterii suplimentare de oprire pentru a preveni creșterea excesivă a arborelui.

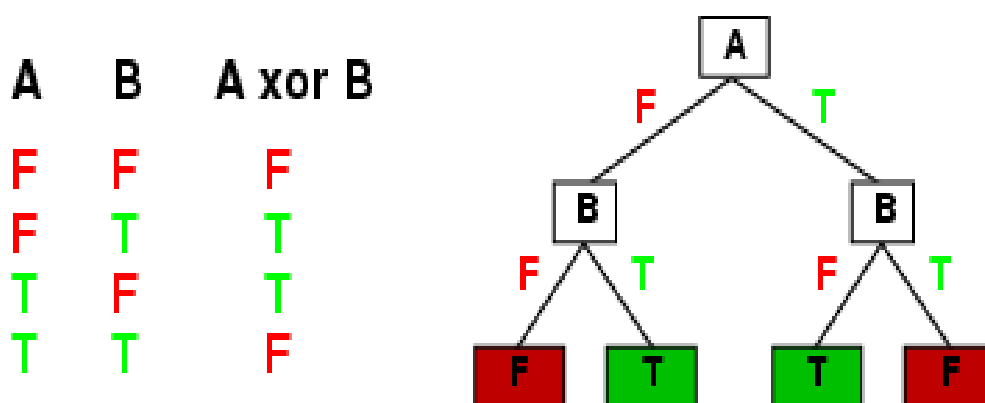


Fig.3.3. Exemplu de expresivitate a unui arbore de decizie pentru funcția XOR

Un exemplu clasic de clasificare binară este problema jocului în funcție de vreme. Datele conțin atribute precum vremea, temperatura, umiditatea și vântul, iar clasa poate fi pozitivă sau negativă. Un arbore de decizie poate învăța reguli de forma: dacă vremea este „nori”, atunci se joacă; dacă vremea este „soare”, se verifică umiditatea; dacă vremea este „ploaie”, se verifică vântul.

Figura 3.4 prezintă un set de date cu 14 instanțe și mai multe atribute meteorologice. Fiecare rând descrie o situație, iar clasa indică rezultatul. Acest tip de tabel permite construirea unui arbore de decizie prin selectarea atributelor care separă cel mai bine clasele pozitive și negative.

No.	Attribute				Clasa
	Vreme	Temperatura	Umiditate	Vant	
1	soare	cald	mare	fals	N
2	soare	cald	mare	adev	N
3	nori	cald	mare	fals	P
4	ploaie	placut	mare	fals	P
5	ploaie	racoare	normal	fals	P
6	ploaie	racoare	normal	adev	N
7	nori	racoare	normal	adev	P
8	soare	placut	mare	fals	N
9	soare	racoare	normal	fals	P
10	ploaie	placut	normal	fals	P
11	soare	placut	normal	adev	P
12	nori	placut	mare	adev	P
13	nori	cald	normal	fals	P
14	ploaie	placut	mare	adev	N

Fig. 3.4. Set de date pentru un exemplu de clasificare binară

Figura 3.5 prezintă arborele rezultat pentru exemplul anterior. Atributul „Vreme” este utilizat ca test inițial, iar ramurile sunt tratate diferit în funcție de valoare. Pentru unele valori se ajunge direct la o clasă, iar pentru altele se folosesc teste suplimentare, cum ar fi „Umiditate” sau „Vânt”. Această reprezentare arată avantajul arborilor de decizie: modelul poate fi transformat ușor în reguli explicabile.

Arborii de decizie pot fi utilizați și în probleme multi-clasă, unde variabila țintă are mai mult de două valori. De exemplu, într-o problemă de evaluare a riscului, clasa poate fi „High”, „Moderate” sau „Low”. În această situație, arborele trebuie să separe exemplele în mai multe clase, nu doar în pozitiv și negativ.

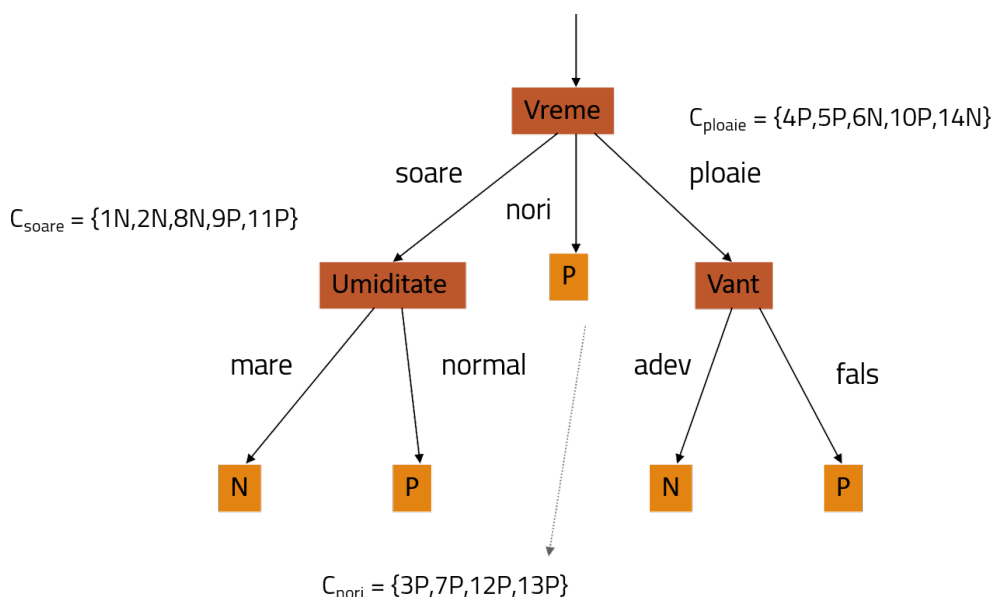


Fig.3.5. Arbore de decizie pentru exemplul de clasificare binară

Nr.	Risc (Clasificare)	Istoric de credit	Datorie	Garanție	Venit
1	Ridicat	Rău	Ridicăță	Niciuna	\$0 până la \$15k
2	Ridicat	Necunoscut	Ridicăță	Niciuna	\$15 până la \$35k
3	Moderat	Necunoscut	Scăzută	Niciuna	\$15 până la \$35k
4	Ridicat	Necunoscut	Scăzută	Niciuna	\$0 până la \$15k
5	Scăzut	Necunoscut	Scăzută	Niciuna	Peste \$35k
6	Scăzut	Necunoscut	Scăzută	Adecvată	Peste \$35k
7	Ridicat	Rău	Scăzută	Niciuna	\$0 până la \$15k
8	Moderat	Rău	Scăzută	Adecvată	Peste \$35k
9	Scăzut	Bun	Scăzută	Niciuna	Peste \$35k
10	Scăzut	Bun	Ridicăță	Adecvată	Peste \$35k
11	Ridicat	Bun	Ridicăță	Niciuna	\$0 până la \$15k
12	Moderat	Bun	Ridicăță	Niciuna	\$15 până la \$35k
13	Scăzut	Bun	Ridicăță	Niciuna	Peste \$35k
14	Ridicat	Rău	Ridicăță	Niciuna	\$15 până la \$35k

Fig. 3.6. Exemplu de set de date pentru o problemă multi-clasă

Figura 3.6 prezintă o problemă de clasificare a riscului, în care clasa poate lua mai multe valori. Atributele includ istoricul de credit, nivelul datoriei, garanțiile și venitul. Spre deosebire de clasificarea binară, aici modelul trebuie să distingă între mai multe categorii de risc.

Figura 3.7 ilustrează modul în care arborele de decizie poate fi extins la probleme multi-clasă. Fiecare frunză indică o clasă posibilă, iar testele intermediare împart progresiv datele după atribute relevante. Astfel, decizia finală este obținută prin parcurgerea arborelui de la rădăcină până la una dintre frunze.

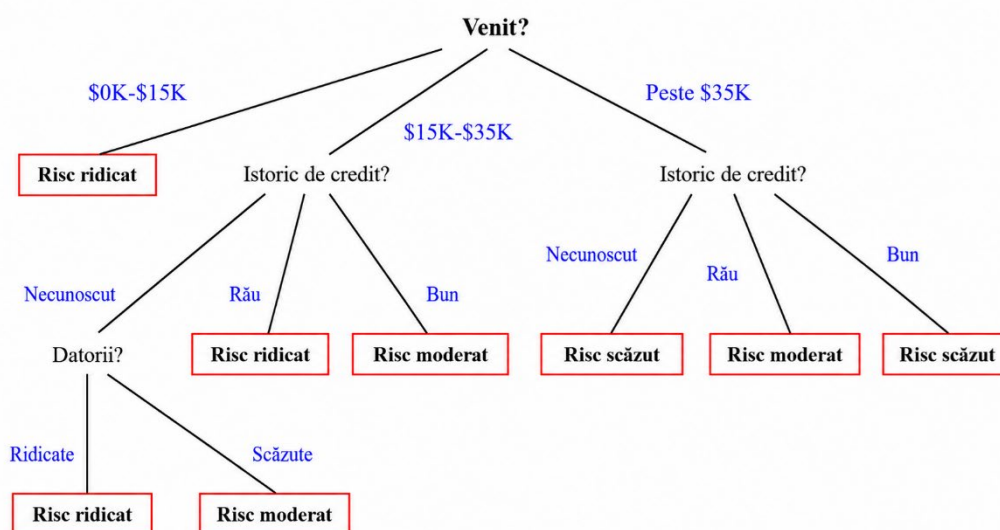


Fig. 3.7. Arbore de decizie pentru o problemă de clasificare multi-clasă

Tipul atributelor influențează modul de partiționare a datelor. Pentru atribute nominale, se poate realiza o partiționare multiplă, în care fiecare valoare distinctă generează o ramură, sau o partiționare binară, în care valorile sunt împărțite în două submulțimi. În cazul atributelor ordinale, valorile au o ordine naturală, iar această ordine poate fi folosită pentru a decide pragurile de separare.

Figura 3.8 prezintă partiționarea unui atribut nominal. Deoarece valorile nu au o ordine naturală, algoritmul poate crea o ramură pentru fiecare valoare sau poate grupa valorile în două submulțimi. Alegerea partiționării influențează forma arborelui și nivelul său de complexitate.

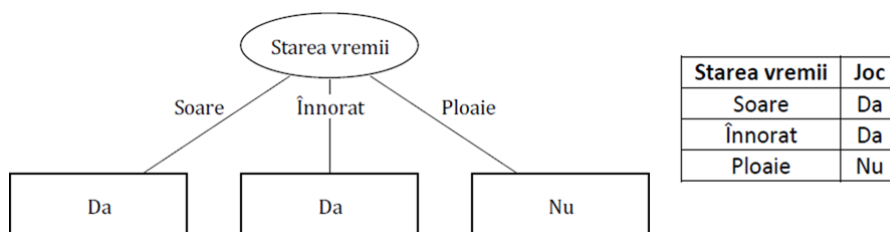


Fig. 3.8. *Partiționarea atributelor nominale într-un arbore de decizie*

Figura 3.9 prezintă cazul atributelor ordinale. Valorile precum mică, medie și mare pot fi ordonate, ceea ce permite construirea unor reguli mai naturale. Totuși, și în acest caz trebuie identificată partiționarea optimă, astfel încât clasele rezultate să fie cât mai bine separate.

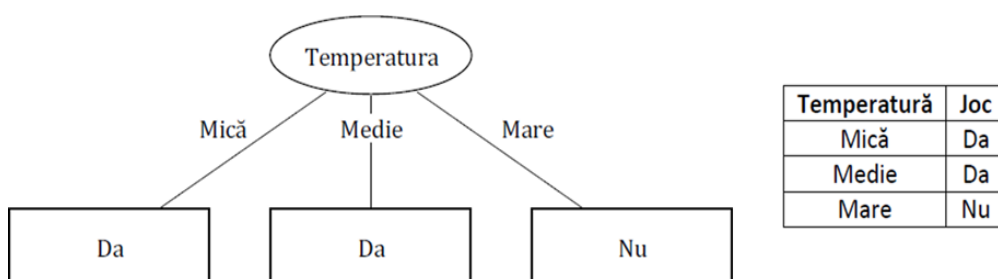


Fig. 3.9. *Partiționarea atributelor ordinale într-un arbore de decizie*

Pentru attribute continue, arborele nu poate testa direct toate valorile posibile. De aceea, datele numerice sunt adesea discretizate. Discretizarea transformă un atribut numeric într-un atribut ordinal prin împărțirea domeniului în intervale. Intervalele pot fi egale ca lățime, pot avea frecvență egală sau pot fi obținute printr-o metodă de grupare. O alternativă este decizia binară de forma $A_i \leq v$, unde v este un prag ales de algoritm.

Figura 3.10 arată discretizarea atributului „Umiditate” în trei intervale egale. Valorile numerice sunt transformate în categorii precum mică, medie și mare. Avantajul este obținerea unei reprezentări mai simple, însă se pierde o parte din informația numerică inițială.

[65, 75], (75, 85], (85, 95]

Umiditate	Joc		Umiditate-Dis1	Joc
65	Da		Mică	Da
70	Da		Mică	Da
72	Da		Mică	Da
75	Da		Mică	Da
80	Da		Medie	Da
85	Da		Medie	Da
86	Nu		Mare	Nu
90	Nu		Mare	Nu
90	Nu		Mare	Nu
91	Nu		Mare	Nu
93	Nu		Mare	Nu
95	Nu		Mare	Nu

Fig. 3.10. Discretizarea unui atribut continuu în intervale egale

Umiditate	Joc		Umiditate	Umiditate-Bin	Joc
65	Da		65	Da	Da
70	Da		70	Da	Da
72	Da		72	Da	Da
75	Da		75	Da	Da
80	Da		80	Da	Da
85	Da		85	Da	Da
86	Nu		86	Nu	Nu
90	Nu		90	Nu	Nu
90	Nu		90	Nu	Nu
91	Nu		91	Nu	Nu
93	Nu		93	Nu	Nu
95	Nu		95	Nu	Nu

$(A_i \leq 85)?$

Fig. 3.11. Discretizarea binară a unui atribut continuu folosind un prag

Figura 3.11 prezintă discretizarea binară a aceluiași atribut folosind pragul 85. Instanțele cu valori mai mici sau egale cu pragul sunt încadrate într-o categorie, iar

celelalte în categoria opusă. Această strategie este foarte des utilizată în arborii de decizie, deoarece permite formularea unor teste simple și eficiente.

Arborii de decizie au mai multe avantaje. Sunt ușor de construit, ușor de implementat și pot clasifica rapid instanțe noi. În plus, sunt interpretabili, deoarece pot fi transformați în reguli de tip *if-then*. Totuși, există și dezavantaje. Antrenarea poate deveni costisitoare pentru seturi mari de date și multe atribute, iar arborii foarte adânci pot deveni excesiv de complecși. În aceste situații, modelul poate avea performanță bună pe datele de antrenare, dar slabă pe date noi.

3.3. Clasificare bazată pe instanțe și algoritmul KNN

Clasificarea bazată pe instanțe pornește de la o idee diferită față de arborii de decizie. În loc să construiască explicit un model sub forma unui arbore sau a unei funcții, algoritmul memorează instanțele de antrenare și le folosește pentru a clasifica instanțele noi. Din acest motiv, astfel de metode sunt numite *lazy learners*, adică metode de învățare „leneșă”. Ele amână cea mai mare parte a calculului până în momentul clasificării unei instanțe noi.

Figura 3.12 arată principiul lazy learning. Instanțele etichetate sunt memorate, iar o instanță nouă este comparată cu exemplele existente. Clasa finală este stabilită pe baza instanțelor de antrenare cele mai apropiate sau mai similare. Această abordare este intuitivă, dar poate deveni costisitoare atunci când setul de date este mare.

Cel mai cunoscut algoritm din această categorie este K-Nearest Neighbors (KNN) [5]. Algoritmul este simplu, dar puternic. El clasifică o instanță nouă pe baza celor mai apropiați K vecini din setul de antrenare. KNN este neparametric, deoarece nu presupune o formă fixă a funcției de decizie, și poate genera limite de decizie flexibile. Totuși, performanța sa depinde foarte mult de alegerea distanței, de scalarea atributelor și de valoarea parametrului K.

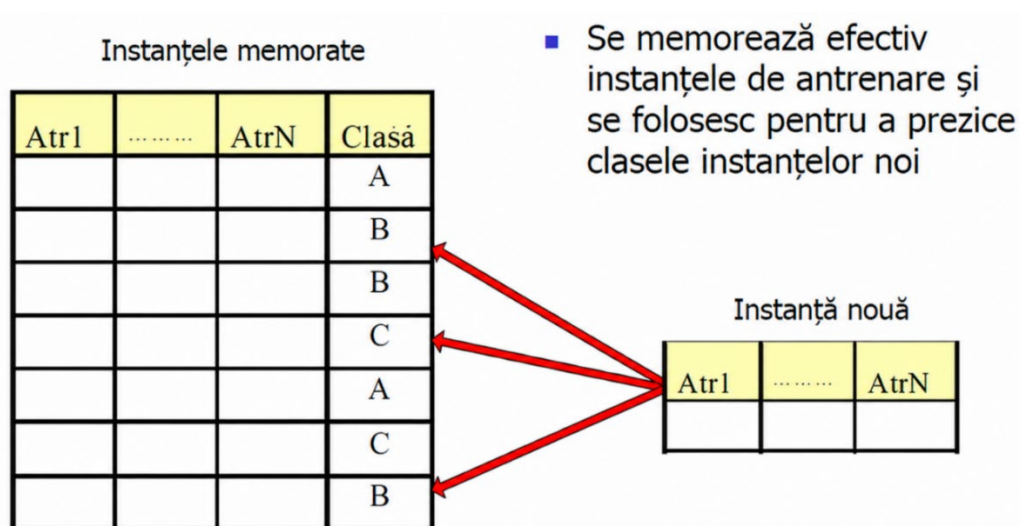


Fig. 3.12. Principiul clasificării bazate pe instanțe

Figura 3.13 ilustrează ideea de bază a algoritmului KNN. Pentru o instanță nouă, se calculează distanța față de instanțele de antrenare, apoi se aleg cele mai apropiate K exemple. Clasificarea se face pe baza acestora. Exemplul de mai sus evidențiază caracterul intuitiv al metodei: noul obiect este atribuit clasei obiectelor cele mai asemănătoare [14].

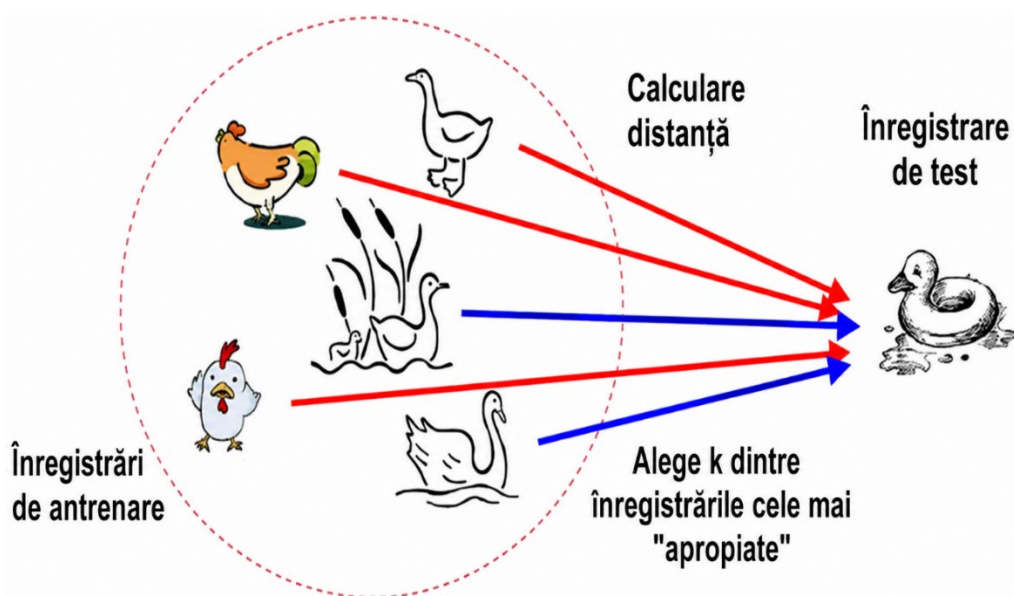


Fig. 3.13. Identificarea vecinilor apropiați în algoritmul KNN

Regula de decizie utilizată cel mai frecvent este votul majorității. Dacă majoritatea celor K vecini aparțin unei clase, atunci instanța nouă primește acea clasă. În problemele multi-clasă, se numără vecinii din fiecare clasă și se alege clasa cu cele mai multe voturi. Există și variante ponderate, în care vecinii mai apropiați au o influență mai mare decât cei mai îndepărtați.

Figura 3.14 prezintă o instanță necunoscută înconjurată de vecini aparținând claselor A și B. Dacă se aleg vecinii din cercul punctat, clasa se stabilește prin votul majorității. Exemplul arată că rezultatul depinde de vecinătatea considerată, deci de valoarea lui K și de măsura de distanță folosită [20].

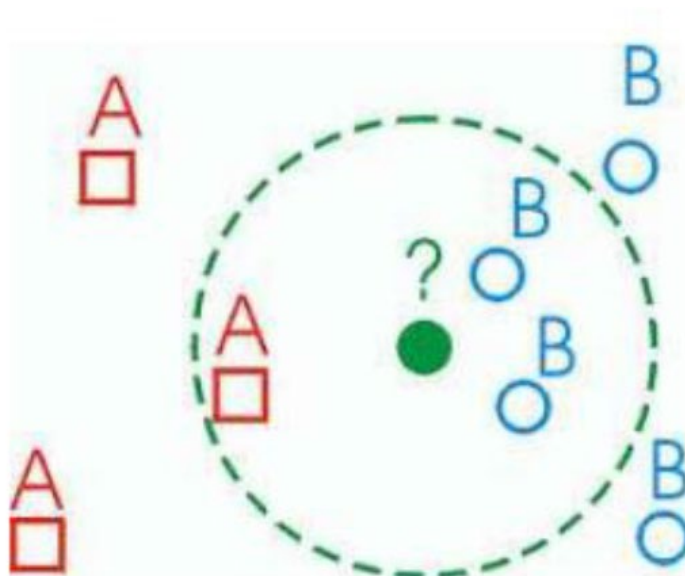


Fig. 3.14. Clasificarea KNN prin votul majorității

Pașii algoritmului KNN sunt următorii:

1. Se alege valoarea lui K .
2. Se calculează distanța dintre instanța de test și fiecare instanță din setul de antrenare.
3. Se sortează distanțele, se selectează cei mai apropiați K vecini.
4. Se determină clasele acestora și se aplică votul majoritar. Pentru a calcula apropierea dintre exemple, se utilizează frecvent distanța euclidiană.

Formula distanței euclidiene permite măsurarea diferenței dintre doi vectori de caracteristici. Într-un spațiu bidimensional, distanța corespunde intuiției geometrice obișnuite, iar în spații cu mai multe dimensiuni formula se extinde prin însumarea diferențelor pătrate pe fiecare atribut.

$$d(x, y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2} \quad (1)$$

O problemă importantă în KNN este scalarea atributelor. Dacă attributele au ordine de mărime diferite, cele cu valori numerice mai mari vor domina calculul distanței. De exemplu, greutatea exprimată în kilograme poate domina înălțimea exprimată în metri, chiar dacă ambele attribute ar trebui să contribuie la decizie. Soluția obișnuită este normalizarea valorilor, de exemplu prin transformarea lor în intervalul $[0, 1]$.

Pentru attribute nominale, calculul distanței nu este la fel de direct ca pentru attribute numerice. De obicei, se poate considera distanța 0 pentru valori identice și 1 pentru valori diferite. Totuși, în anumite aplicații pot exista distanțe mai rafinate: de exemplu, într-un spațiu al culorilor, galbenul poate fi considerat mai apropiat de portocaliu decât de verde. Uneori, attributele pot primi ponderi diferite, atunci când unele sunt considerate mai importante decât altele.

Alegerea parametrului K are un impact major asupra performanței. Dacă K este prea mic, modelul devine sensibil la zgomot și poate produce overfitting. De exemplu, pentru $K = 1$, o singură instanță de antrenare greșită poate schimba complet decizia. Dacă K este prea mare, vecinătatea poate include puncte îndepărtate, nerelevante pentru instanța curentă, iar modelul poate deveni prea general.

Figura 3.15 arată cum se modifică vecinătatea atunci când K crește. Pentru $K = 1$ se ia în considerare doar cel mai apropiat vecin, în timp ce pentru valori mai mari cercul de vecinătate include mai multe puncte. Alegerea trebuie să balanseze capacitatea de a captura detalii locale cu obținerea unor decizii stabile.

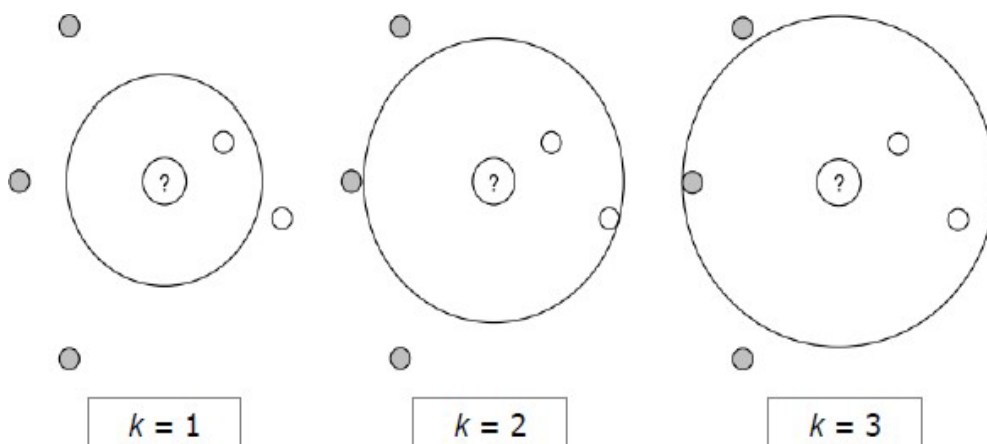


Fig. 3.15. Influența valorii parametrului K asupra vecinătății considerate

Avantajele KNN sunt simplitatea, interpretabilitatea și flexibilitatea limitelor de decizie. Algoritmul este ușor de implementat și nu presupune antrenare complexă. Dezavantajele apar în special pentru seturi de date mari sau cu dimensionalitate ridicată. Clasificarea poate fi costisitoare, deoarece trebuie calculate distanțe față de multe exemple. În plus, atributele irelevante sau corelate pot afecta puternic rezultatul, motiv pentru care selecția caracteristicilor și scalarea sunt etape importante.

3.4. Support Vector Machine (SVM)

Support Vector Machine este un algoritm utilizat în special pentru sarcini de clasificare. Ideea principală este identificarea unui hiperplan optim care separă punctele din clase diferite. Într-un spațiu cu două caracteristici, hiperplanul este o linie. Într-un spațiu cu trei caracteristici, el este un plan. În general, pentru un spațiu cu mai multe dimensiuni, hiperplanul este o suprafață de decizie de dimensiune corespunzătoare.

SVM nu caută doar o separare oarecare între clase, ci separarea cu marjă maximă. Marja reprezintă distanța dintre hiperplan și cele mai apropiate puncte din fiecare clasă. Hiperplanul optim este cel care maximizează această distanță, deoarece o marjă mai mare tinde să ofere o capacitate mai bună de generalizare. Punctele care se află cel mai aproape de hiperplan se numesc vectori suport.

Figura 3.16 prezintă mai multe benzi de decizie posibile pentru separarea a două clase. SVM urmărește alegerea suprafeței care separă clasele la distanță maximă de punctele cele mai apropiate. Această idee diferențiază SVM de metodele care se limitează la orice separator corect pe datele de antrenare.

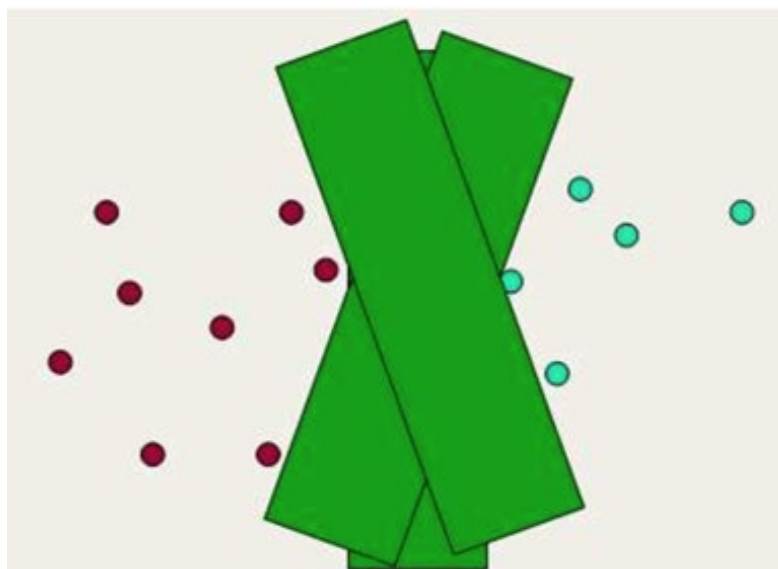


Fig. 3.16. Suprafețe de decizie posibile în SVM

Figura 3.17 arată că pot exista mai multe linii capabile să separe două clase, însă nu toate sunt la fel de bune. Linia optimă este cea care maximizează marja, deoarece este mai puțin sensibilă la mici variații ale datelor. În cazul unui spațiu bidimensional, hiperplanul este reprezentat printr-o dreaptă.

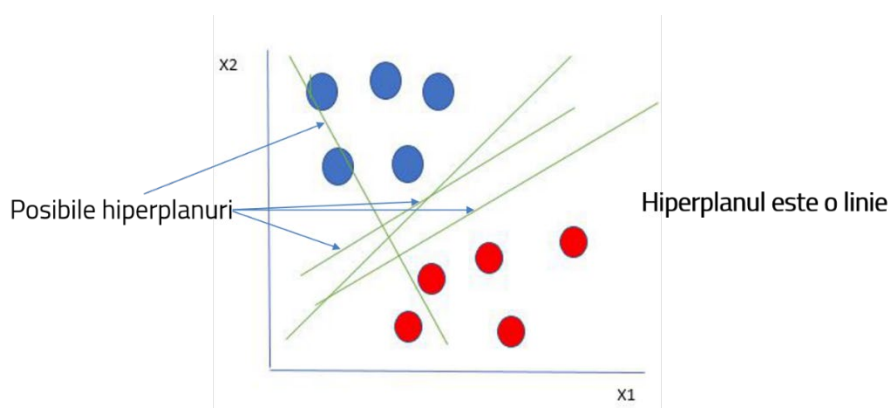


Fig. 3.17. Posibile hiperplanuri pentru separarea a două clase în 2D

În cazul 2D, separatorul poate fi descris printr-o ecuație liniară de forma $ax + by = c$. Punctele aflate de o parte a liniei aparțin unei clase, iar punctele aflate de cealaltă parte aparțin celeilalte clase. Clasificarea unui punct nou se face în funcție de semnul expresiei asociate hiperplanului. Această formulare geometrică oferă o interpretare clară a deciziei.

Figura 3.18 prezintă o problemă 2D în care două grupuri de puncte pot fi separate printr-o linie. Ecuația $ax + by \geq c$ poate caracteriza una dintre clase, iar $ax + by < c$ cealaltă clasă. În acest caz, modelul este liniar, deoarece suprafața de separare este o dreaptă.

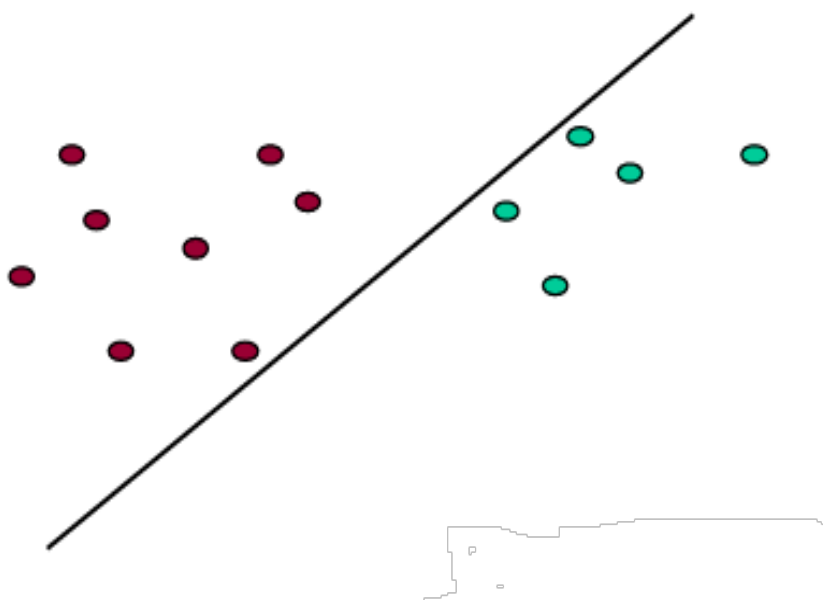


Fig. 3.18. Separarea liniară a două clase într-un spațiu bidimensional

Figura 3.19 definește cei mai importanți termeni ai metodei SVM. Vectorii suport sunt punctele cele mai apropiate de hiperplan, iar marja este distanța dintre hiperplan și aceste puncte. Acești vectori suport au un rol critic, deoarece poziția lor determină direct forma separatorului optim.

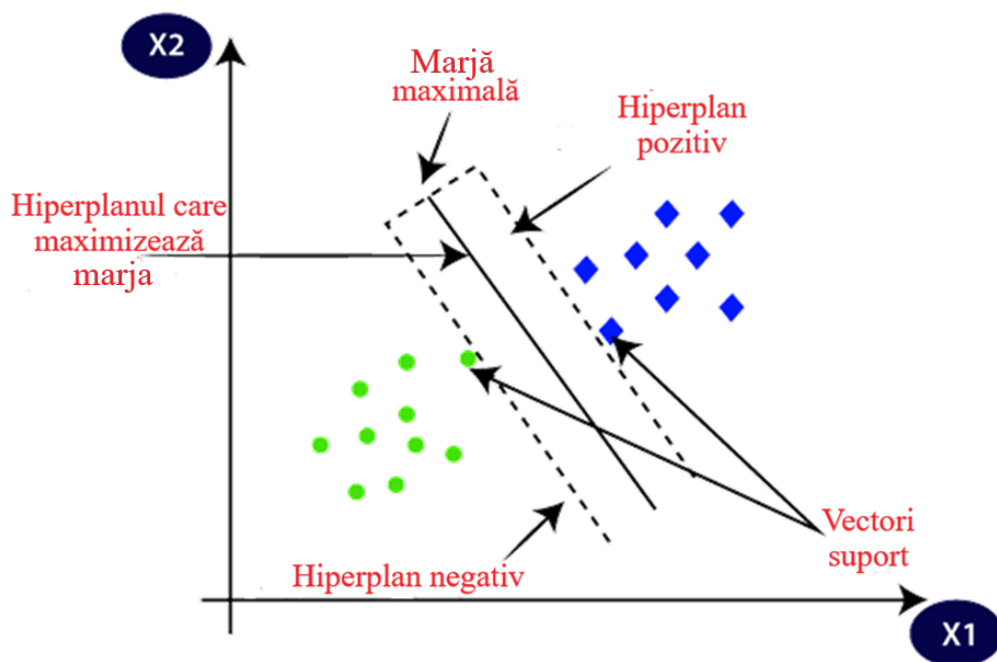


Fig. 3.19. Vectori suport și marjă în SVM

Formal, pentru o problemă de clasificare binară, se consideră un set de date format din vectori de caracteristici X și etichete Y , unde etichetele pot fi $+1$ și -1 . Ecuația hiperplanului poate fi scrisă $w^T x + b = 0$, unde w este vectorul normal la hiperplan, iar b este termenul de deplasare. Semnul expresiei $w^T x + b$ indică de ce parte a hiperplanului se află punctul.

$$y = \begin{cases} 1 : w^T x + b \geq 0 \\ 0 : w^T x + b < 0 \end{cases} \quad (2)$$

Figura 3.20 detaliază cazul SVM liniar. Pentru vectorii suport, inegalitățile devin egalități, iar marja este determinată de distanța față de hiperplan. Obiectivul algoritmului este maximizarea acestei marje, ceea ce echivalează cu alegerea unui separator cât mai robust.

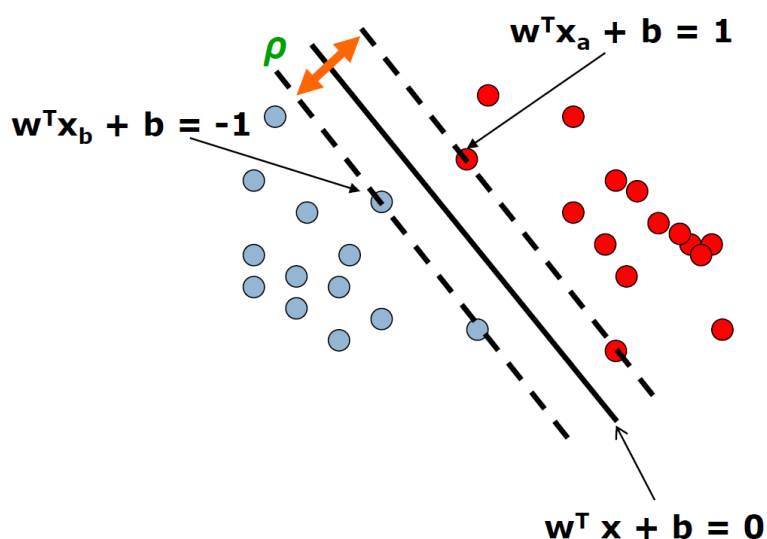


Fig. 3.20. Constrângeri și marjă pentru SVM liniar

După antrenare, clasificarea unui punct nou se face prin calculul scorului față de hiperplan. Dacă scorul este pozitiv, punctul este atribuit unei clase; dacă este negativ, celeilalte clase. În unele aplicații se poate folosi și un prag de încredere. Dacă scorul este foarte apropiat de zero, decizia poate fi considerată incertă, deoarece punctul se află aproape de suprafața de separare.

Figura 3.21 prezintă modul în care un punct nou este proiectat în raport cu normala hiperplanului. Scorul rezultat indică poziția punctului față de suprafața de decizie. O valoare pozitivă sau negativă determină clasa, iar valorile apropiate de zero pot indica un nivel mai redus de încredere în predicție.

Score $> t$: da
 Score $< -t$: nu
 Altfel: ?

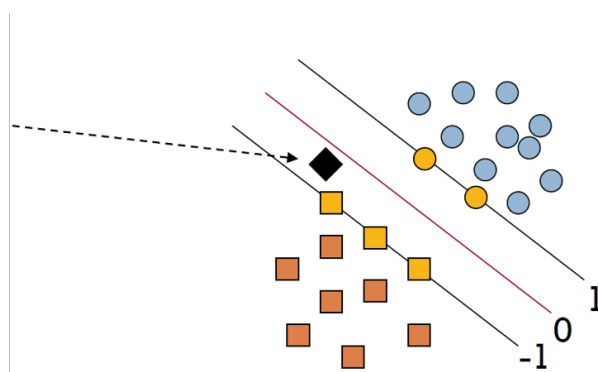


Fig. 3.21. Clasificarea unui punct nou cu ajutorul scorului SVM

Figura 3.22 prezintă un exemplu în care două clase, reprezentate prin puncte verzi și albastre, trebuie separate printr-o frontieră de decizie. SVM caută separatorul care oferă cea mai bună separare geometrică, ținând cont de punctele aflate cel mai aproape de frontieră.

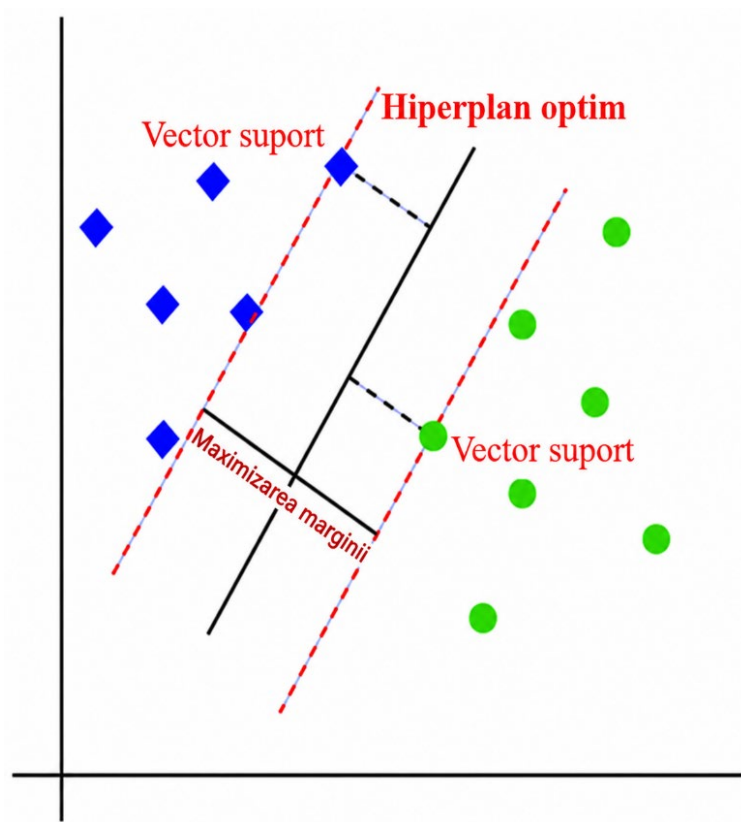


Fig. 3.22. Exemplu de clasificare SVM cu două clase

SVM este eficient în probleme în care datele pot fi separate aproximativ printr-o frontieră clară. În varianta liniară, frontiera este un hiperplan. În variante mai avansate, prin metode kernel, datele pot fi proiectate implicit într-un spațiu de dimensiune mai mare, unde separarea devine posibilă. În această secțiune accentul este pus pe interpretarea liniară, deoarece aceasta oferă intuiția geometrică fundamentală.

3.5. Evaluarea modelelor de clasificare

Un model de clasificare nu trebuie evaluat doar prin cât de bine se potrivește pe setul de antrenare. Scopul real este performanța pe date noi. De aceea, se face diferența între eroarea de antrenare și eroarea de testare. Eroarea de antrenare măsoară cât de bine explică modelul datele pe care le-a văzut în timpul antrenării. Eroarea de testare estimează capacitatea de generalizare.

Figura 3.23 prezintă diferența dintre underfitting și overfitting. Un model cu bias ridicat și varianță redusă poate fi prea simplu și nu reușește să învețe structura datelor. Un model cu bias redus și varianță ridicată poate memora exemplele de antrenare, inclusiv zgomotul, ceea ce conduce la performanțe slabe pe date noi.

Evaluarea clasificatorilor se realizează prin măsuri statistice. Cele mai utilizate sunt acuratețea, precizia, recall-ul și scorul F1. Alegerea metricii depinde de problema analizată. În unele situații, acuratețea este suficientă. În altele, mai ales când clasele sunt dezechilibrate, precizia și recall-ul oferă o imagine mai relevantă asupra comportamentului modelului [8].

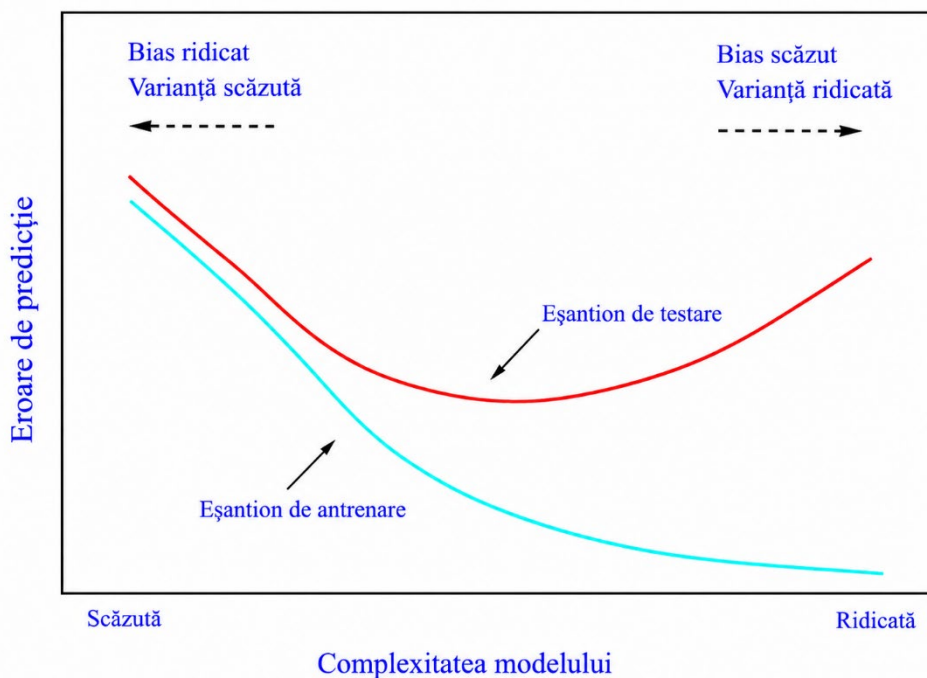


Fig. 3.23. Relația dintre eroarea de antrenare, eroarea de testare, overfitting și underfitting

Există două direcții de evaluare: măsuri statistice și eficiență. Măsurile statistice indică nivelul de corectitudine al modelului, iar eficiența se referă la resursele necesare pentru dezvoltarea și testarea modelului. În practică, un model trebuie să fie nu doar precis, ci și suficient de rapid și robust pentru aplicația vizată.

Acuratețea este raportul dintre numărul de exemple corect clasificate și numărul total de exemple. Este opusul erorii de clasificare și poate fi calculată pe setul de validare sau pe setul de testare. Deși este ușor de interpretat, acuratețea poate fi înșelătoare atunci când clasele sunt dezechilibrate. De exemplu, dacă 95% dintre exemple aparțin unei clase, un model care prezice mereu acea clasă poate avea acuratețe mare, dar utilitate redusă.

Precizia măsoară, dintre toate exemplele clasificate ca pozitive, câte sunt într-adevăr pozitive. Precizia este importantă atunci când erorile de tip fals pozitiv sunt costisitoare. Recall-ul determină proporția exemplelor pozitive reale care au fost identificate corect de clasificator. Este relevant atunci când este important să nu ratăm exemple pozitive.

Scorul F1 combină precizia și recall-ul prin media armonică, oferind o singură valoare utilă pentru compararea modelelor. Acest indicator este important mai ales în probleme cu dezechilibru între clase sau cu compromisuri între fals pozitiv și fals negativ.

O reprezentare utilă pentru evaluarea clasificatorilor este matricea de confuzie. Aceasta separă exemplele în patru categorii: adevărate pozitive, false pozitive, adevărate negative și false negative. Pe baza acestei matrice se pot calcula acuratețea, precizia, recall-ul și scorul F1. Matricea de confuzie este mai informativă decât o singură valoare agregată, deoarece arată tipurile concrete de erori comise de model.

Figura 3.24 sintetizează metricile de performanță utilizate în clasificare. Elementele din matricea de confuzie permit calcularea acurateței, preciziei, recall-ului și scorului F1. În analiza modelelor, aceste valori trebuie interpretate împreună, nu izolat, deoarece fiecare surprinde o parte diferită a comportamentului clasificatorului.

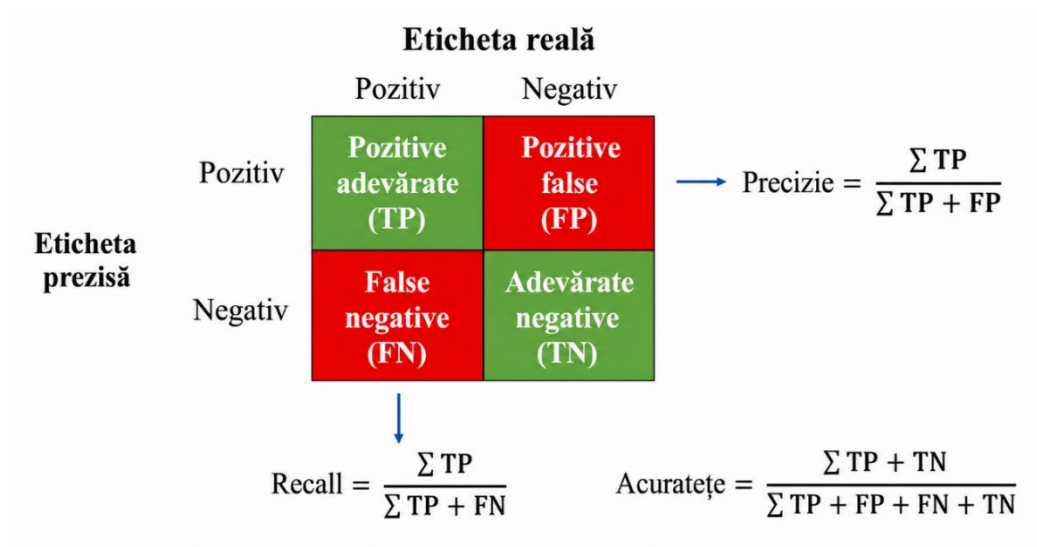


Fig. 3.24. Matrici de performanță calculate pe baza matricei de confuzie

Așadar, clasificarea este una dintre cele mai importante sarcini ale învățării supervizate. Ea presupune învățarea unei relații între atributele de intrare și o clasă discretă. Arborii de decizie oferă modele ușor de interpretat, KNN clasifică prin similaritate față de exemplele existente, iar SVM caută o frontieră de decizie cu marjă maximă. Indiferent de algoritm, performanța trebuie evaluată prin capacitatea de generalizare, nu doar prin ajustarea modelului pe datele de antrenare.

4. Capitolul 4. Tehnici de învățare supervizată: Regresie.

Capitolul prezintă a doua categorie importantă de tehnici de învățare supervizată, respectiv regresia. Spre deosebire de clasificare, unde modelul trebuie să aleagă o etichetă dintr-o mulțime finită de clase, regresia urmărește prezicerea unei valori continue. Din acest motiv, regresia este utilizată frecvent în probleme de estimare, prognoză, modelare a relațiilor dintre variabile și analiză a tendințelor observate în date.

În cadrul acestui capitol sunt introduse noțiunile de corelație, aproximare a unei funcții, regresie liniară simplă și multiplă, funcții de cost, indicatori de evaluare și regresie logistică. Deși regresia logistică este denumită „regresie”, ea este folosită în mod obișnuit pentru probleme de clasificare binară, deoarece estimează probabilitatea apartenenței unei instanțe la o anumită clasă. Astfel, capitolul realizează o legătură între estimarea numerică specifică regresiei și decizia de clasificare bazată pe probabilități.

4.1. Introducere în Regresie

În învățarea supervizată, datele de antrenare conțin atât exemple de intrare, cât și valorile corecte ale ieșirii. Dacă ieșirea este discretă, problema este una de clasificare. Dacă ieșirea este numerică și continuă, problema este una de regresie. Această diferență este esențială, deoarece determină atât tipul de algoritm ales, cât și modul în care este evaluată performanța modelului.

Așa cum am văzut în capitolul anterior, clasificarea presupune învățarea unei funcții discrete. De exemplu, un sistem poate decide dacă un e-mail este spam sau non-spam, dacă o imagine conține un anumit obiect sau dacă o tranzacție este frauduloasă. Regresia, în schimb, urmărește învățarea unei funcții continue. Modelul nu returnează o clasă, ci o valoare numerică: temperatura estimată pentru ziua următoare, consumul de energie, prețul unui produs, durata estimată a unui proces sau orice altă mărime care poate fi exprimată numeric.

Regresia este o tehnică de învățare supervizată deoarece modelul este construit pe baza unor exemple pentru care valoarea țintă este cunoscută. Fiecare exemplu conține una sau mai multe variabile independente, denumite și caracteristici, și o variabilă

dependentă, denumită variabilă țintă. Scopul algoritmului este să identifice relația dintre caracteristici și țintă, astfel încât să poată estima valoarea țintă pentru instanțe noi.

În practică, regresia poate fi utilizată pentru predicție, prognoză, modelarea seriilor de timp sau studierea relațiilor dintre variabile. Într-un context tehnic, aceasta poate fi folosită pentru estimarea consumului unui sistem, pentru modelarea comportamentului unui senzor, pentru predicția unei valori măsurate sau pentru identificarea unei tendințe în date experimentale. În toate aceste cazuri, scopul principal este obținerea unei valori numerice cât mai apropiate de valoarea reală.

Înainte de construirea unui model de regresie, este important să fie analizată relația dintre variabile. Noțiunea de corelație descrie gradul în care două variabile se modifică împreună. Dacă valorile unei variabile cresc atunci când cresc și valorile celeilalte variabile, se poate vorbi despre o corelație pozitivă. Dacă valorile unei variabile cresc atunci când cealaltă scade, relația este negativă. Dacă nu se observă o asociere liniară clară, corelația poate fi apropiată de zero.

Coeficientul de corelație Pearson este una dintre cele mai cunoscute măsuri ale dependenței liniare dintre două variabile. Acesta ia valori între -1 și $+1$. O valoare apropiată de $+1$ indică o relație liniară pozitivă puternică, o valoare apropiată de -1 indică o relație liniară negativă puternică, iar o valoare apropiată de 0 indică lipsa unei relații liniare evidente. Totuși, corelația trebuie interpretată cu atenție: existența unei corelații nu demonstrează automat o relație cauzală.

Regresia poate fi înțeleasă intuitiv ca o problemă de aproximare a unei funcții. Având un set de puncte observate, se caută o funcție care descrie cât mai bine relația dintre intrări și ieșire. În cazul cel mai simplu, această funcție este o linie dreaptă. În situații mai complexe, funcția poate fi neliniară sau poate depinde de mai multe variabile. Ideea centrală rămâne aceeași: modelul trebuie să ofere o aproximare fidelă a fenomenului observat.

Figura 4.1 ilustrează problema de bază a regresiei: se observă mai multe puncte într-un sistem de coordonate și se dorește identificarea unei funcții care să treacă cât mai aproape de acestea. În general, nu există o singură funcție evidentă care să modeleze perfect toate punctele, deoarece datele reale conțin variații, erori de măsurare sau zgomot. De aceea, regresia caută o funcție care minimizează eroarea globală de aproximare.

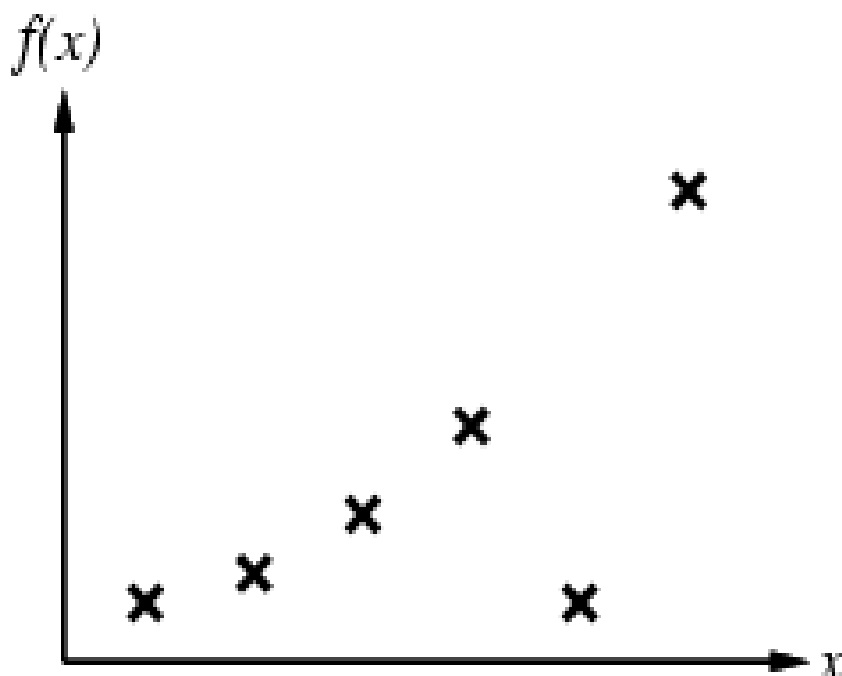


Fig. 4.1. *Exemplu intuitiv de aproximare a unei funcții pe baza unor puncte observate*

Pentru același set de date pot exista mai multe funcții posibile. Unele pot fi foarte simple, cum ar fi o linie dreaptă, iar altele pot fi foarte complexe, urmărind aproape fiecare variație din datele de antrenare. Alegerea modelului trebuie făcută astfel încât acesta să surprindă relația importantă dintre variabile, fără să memoreze zgomotul.

Figura 4.2 evidențiază faptul că o funcție prea simplă poate rata structura datelor, iar o funcție prea complexă poate ajunge să urmărească fluctuații nerelevante. În învățarea automată, această problemă este asociată cu echilibrul dintre sub-potrivire și supra-potrivire. Modelul trebuie să fie suficient de expresiv pentru a surprinde relația reală, dar nu atât de flexibil încât să devină dependent de particularitățile irelevante ale setului de antrenare.

Principiul cunoscut sub denumirea de Briciul lui Occam (*Entia non sunt multiplicanda praeter necessitatem*) afirmă că, atunci când mai multe ipoteze explică datele la fel de bine, trebuie preferată ipoteza cea mai simplă. În regresie, acest principiu se traduce prin alegerea unui model care oferă un compromis bun între simplitate și acuratețe. Modelele mai simple sunt mai ușor de interpretat și tind să generalizeze mai bine atunci când datele disponibile sunt limitate.

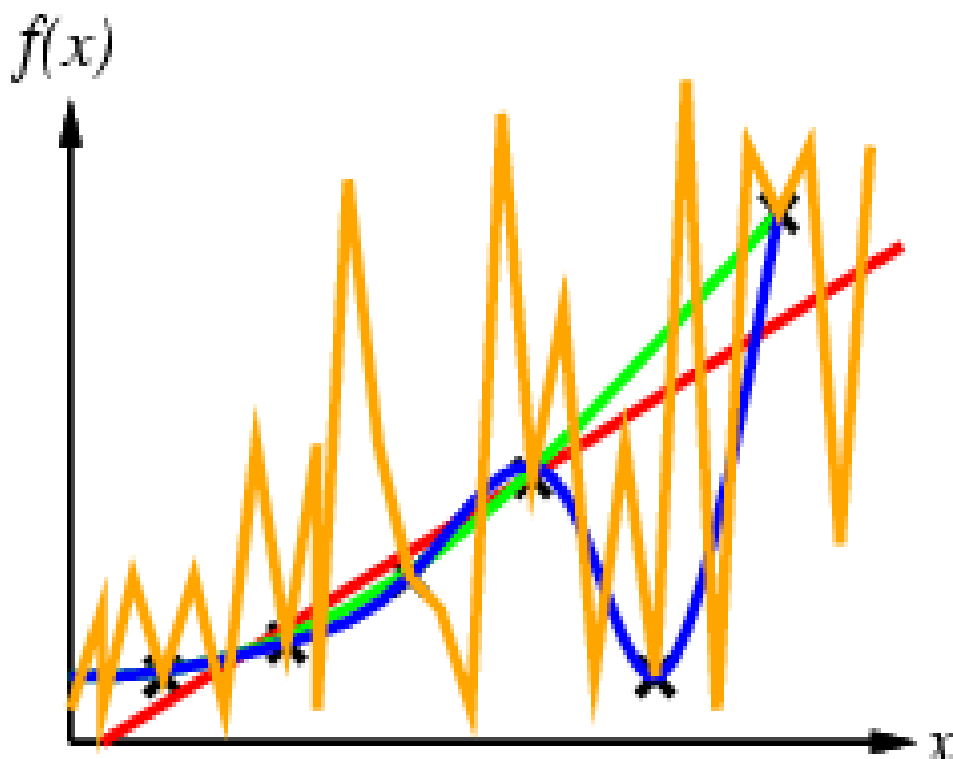


Fig. 4.2. Modalități diferite de aproximare a aceluiași set de puncte

Astfel, potrivit principiului de parcimonie în contextual aproximării datelor – o linie dreaptă poate reprezenta uneori o explicație suficientă, chiar dacă nu trece exact prin toate punctele. O curbă extrem de neregulată poate produce o eroare mai mică pe datele de antrenare, dar riscă să devină inutilă pentru date noi. Din această perspectivă, obiectivul regresiei nu este memorarea punctelor, ci identificarea unei relații care să rămână validă și în afara setului utilizat pentru antrenare.

În regresie, un model care generalizează bine are erori mici pe date noi, nu doar pe datele cunoscute. Pentru verificarea acestei capacități, setul de date este împărțit de regulă în cel puțin două subseturi: o mulțime de antrenare, folosită pentru construirea modelului, și o mulțime de testare sau validare, folosită pentru evaluarea performanței.

Sub-potrivirea apare atunci când modelul este prea simplu și nu poate învăța distribuția datelor. În acest caz, atât eroarea de antrenare, cât și eroarea de testare sunt mari. Supra-potrivirea apare atunci când modelul este prea complex și învață nu doar structura utilă a datelor, ci și zgomotul. În această situație, eroarea pe datele de antrenare poate fi foarte mică, dar performanța pe date noi este slabă.

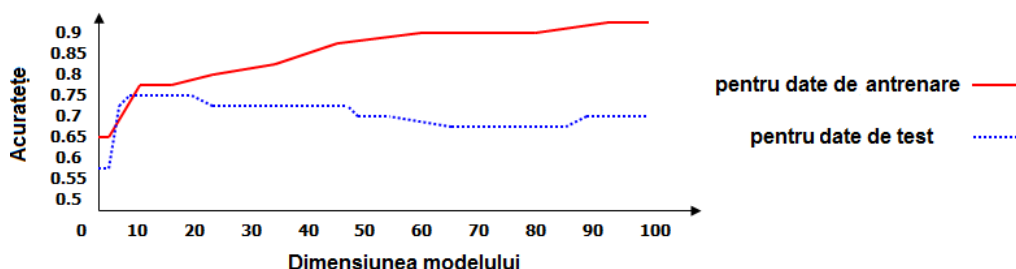


Fig. 4.3. Exemplu de model supra-potrivit

Figura 4,3 ilustrează un model supra-potrivit, care se adaptează excesiv la punctele de antrenare și poate produce predicții nesigure pentru date noi. Un model subpotrivit nu surprinde tendința datelor, iar un model bine ales urmează structura generală, fără să urmărească fiecare fluctuație.

4.2. Regresie liniară simplă și multiplă

Regresia liniară este una dintre cele mai utilizate metode de regresie, datorită simplității sale și a interpretabilității ridicate. Ideea de bază este determinarea unei relații liniare între o variabilă dependentă și una sau mai multe variabile independente. Dacă există o singură variabilă independentă, se vorbește despre regresie liniară simplă. Dacă există două sau mai multe variabile independente, se poate aplica regresia liniară multiplă.

În regresia liniară simplă, modelul poate fi scris sub forma:

$$\hat{y} = a + b * x , \quad (3)$$

unde : $\hat{y}$ este valoarea estimată a variabilei dependente ; x este variabila independentă ;
 a este interceptarea, iar b este panta dreptei.

Interceptarea indică valoarea estimată a lui y atunci când x este zero, iar panta arată cât se modifică y atunci când x crește cu o unitate. Dacă b este pozitiv, relația este crescătoare; iar dacă b este negativ, relația este descrescătoare. În cazul în care b este apropiat de zero, variabila x are o influență liniară redusă asupra lui y .

Figura 4.4 prezintă ideea unei relații liniare între o variabilă independentă și o variabilă dependentă. Punctele reprezintă observațiile, iar linia de regresie ilustrează modelul care aproximează tendința generală. Scopul modelului este să poziționeze această linie astfel încât diferențele dintre valorile reale și cele estimate să fie cât mai mici.

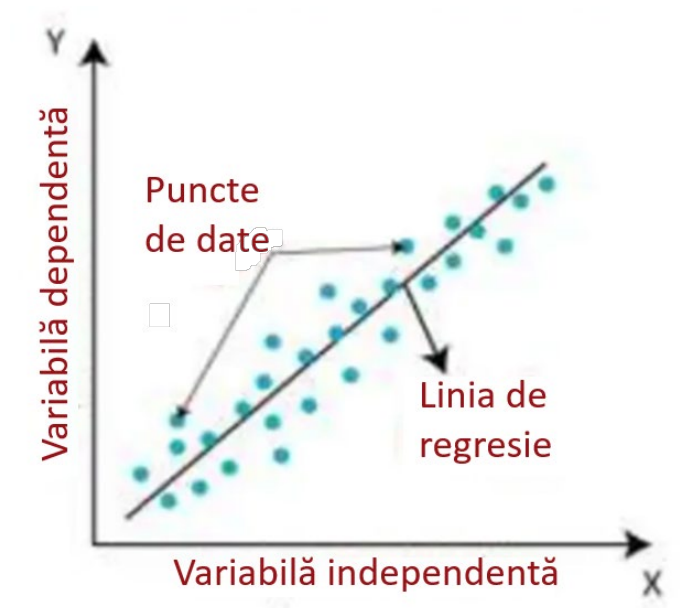
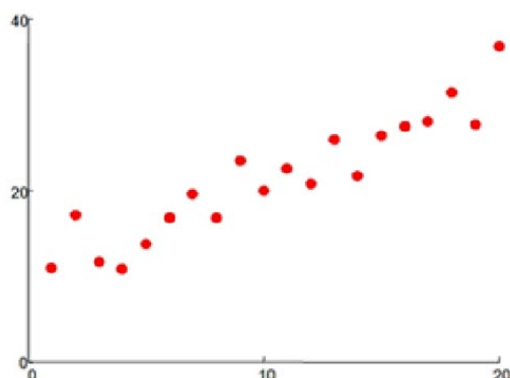


Fig. 4.4. Regresia liniară ca metodă de determinare a relației dintre variabile

Regresia liniară poate fi extinsă natural la mai multe variabile independente. În acest caz, modelul nu mai descrie o linie într-un plan, ci un plan sau un hiperplan într-un spațiu cu mai multe dimensiuni. Pentru două variabile independente, modelul poate fi reprezentat printr-o suprafață. Pentru mai mult de două variabile, reprezentarea grafică devine dificilă, dar principiul matematic rămâne același.

Figura 4.5 arată diferența dintre regresia liniară simplă și regresia liniară cu două variabile independente. În primul caz, relația este ilustrată printr-o linie în plan. În al doilea caz, valorile sunt reprezentate într-un spațiu tridimensional, iar modelul devine o suprafață care estimează variabila dependentă pe baza celor doi predictorii [25].

O variabilă independentă



2 variabile independente

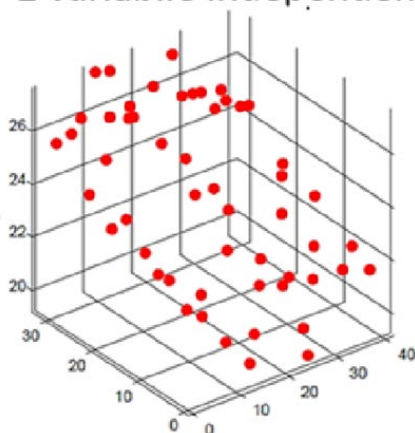


Fig. 4.5. Reprezentarea regresiei liniare în cazul unei variabile și în cazul a două variabile independente

Regresia liniară simplă este potrivită atunci când se dorește analizarea relației dintre două variabile. De exemplu, se poate studia influența înălțimii asupra greutateii, relația dintre timpul de studiu și nota obținută sau relația dintre dimensiunea unui produs și prețul său. Regresia multiplă este utilizată atunci când fenomenul analizat depinde de mai mulți factori. Într-un astfel de caz, fiecare predictor contribuie la estimarea valorii țintă.

Figura 4.6 prezintă două situații distincte. În regresia liniară simplă, salariul poate fi estimat pe baza duratei muncii, folosind o singură caracteristică.

Regresie liniară simplă



Regresie liniară multiplă

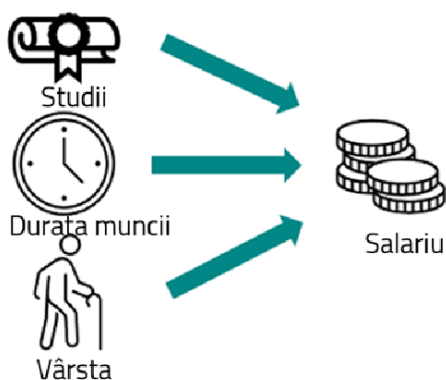


Fig. 4.6. Diferența dintre regresia liniară simplă și regresia liniară multiplă

În regresia liniară multiplă, salariul poate fi estimat pe baza mai multor variabile, cum ar fi durata muncii, vârsta sau nivelul studiilor. Cu cât se folosesc mai mulți predictorii relevanți, cu atât modelul poate descrie mai bine fenomenul analizat, însă crește și riscul de complexitate inutilă sau de includere a unor variabile irelevante.

În cazul regresiei liniare simple, obiectivul este determinarea dreptei care descrie cel mai bine relația dintre variabila independentă X și variabila dependentă y . Dacă punctele sunt apropiate de o linie dreaptă, relația liniară este puternică și modelul poate produce predicții bune. Dacă punctele sunt foarte dispersate, regresia liniară poate să nu fie suficientă pentru a descrie fenomenul.

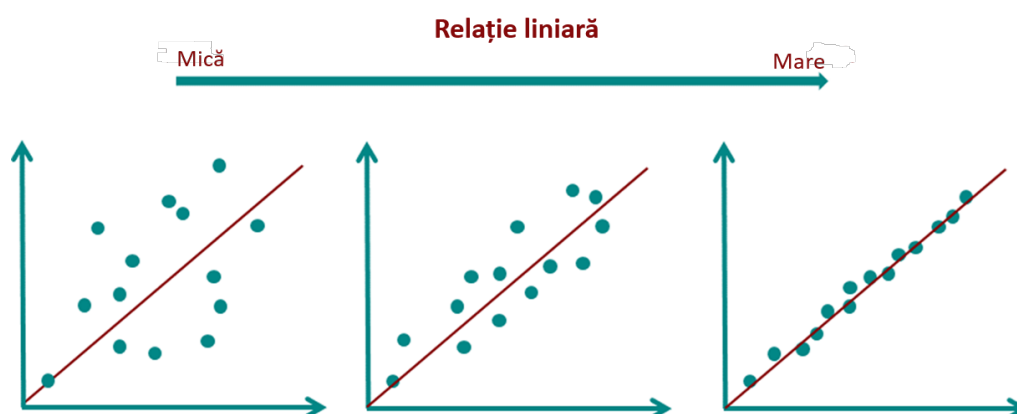


Fig. 4.7. Exemplu de regresie liniară simplă: estimarea greutății în funcție de înălțime

Figura 4.7 prezintă evoluția unei relații liniare. În primul caz, dispersia ridicată a observațiilor în jurul dreptei de regresie indică o corelație redusă și erori de estimare mari. În al doilea caz, dispersia scade, ceea ce reflectă o relație liniară mai pronunțată și o precizie mai bună a modelului. În ultimul caz, punctele sunt aproape perfect aliniate pe dreapta de regresie, indicând o relație liniară foarte puternică și erori de estimare aproape inexistente. Acest exemplu evidențiază faptul că apropierea observațiilor de dreapta de regresie reprezintă un indicator al calității modelului linear.

Modelul regresiei liniare simple poate fi completat cu un termen de eroare:

$$y = a + b * x + \varepsilon \quad (4)$$

Termenul $\mathcal{E}$ descrie diferența dintre valoarea reală și valoarea estimată de model. În datele reale, această diferență apare aproape întotdeauna, deoarece relațiile dintre variabile nu sunt perfecte, iar măsurătorile pot conține zgomot. Rolul algoritmului este să determine parametrii a și b astfel încât erorile să fie cât mai mici în ansamblu.

Un exemplu simplu poate fi estimarea prețului unei pizza în funcție de diametrul acesteia. Dacă există o relație relativ liniară între dimensiune și preț, modelul poate fi folosit pentru a prezice prețul unei pizza de dimensiune necunoscută. Chiar dacă exemplul este simplu, el evidențiază modul în care regresia transformă observațiile din trecut într-un instrument de predicție.

Figura 4.8 prezintă un set de valori care poate fi folosit pentru construirea unui model liniar. Fiecare punct reprezintă o observație de forma diametru - preț. După determinarea parametrilor, modelul poate fi utilizat pentru a estima prețul unei pizza cu diametrul de 20 inch, pe baza relației învățate din date.

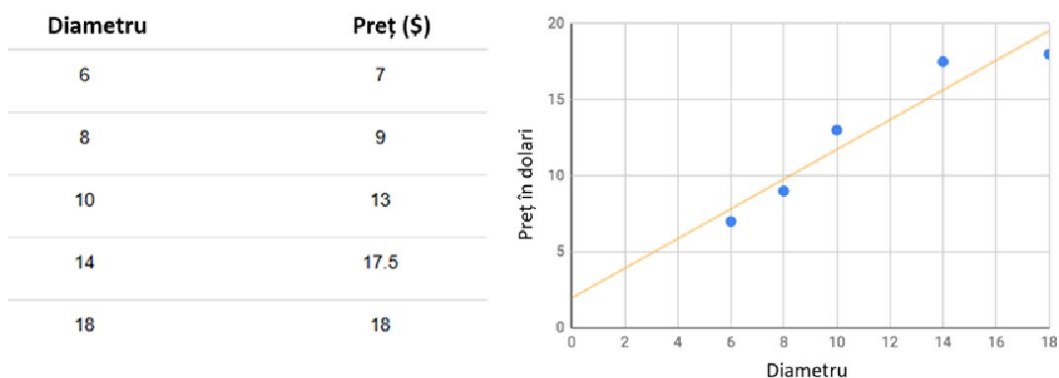


Fig. 4.8. Exemplu de predicție a prețului unei pizza pe baza dimensiunii

Coeficienții regresiei pot fi calculați pe baza sumelor valorilor observate. Pentru regresia liniară simplă, panta și interceptarea se determină astfel încât suma erorilor pătratice să fie minimă. Această abordare este cunoscută sub denumirea de metoda celor mai mici pătrate. Ea alege linia care produce cea mai mică eroare totală, ținând cont de toate punctele din setul de date.

x	y	x ²	xy
3	8	9	24
9	6	81	54
5	4	25	20
3	2	9	6
$\Sigma x = 20$	$\Sigma y = 20$	$\Sigma x^2 = 124$	$\Sigma xy = 104$

x	y
3	8
9	6
5	4
3	2

$$a = \frac{\sum y \sum x^2 - \sum x \sum xy}{n \sum x^2 - (\sum x)^2} = \frac{20 * 124 - 20 * 104}{4 * 124 - 20^2} = \frac{400}{96} = 4.17$$

$$b = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2} = \frac{4 * 104 - 20 * 20}{4 * 124 - 20^2} = \frac{16}{96} = 0.166$$

$$y = 4.17 + 0.166x$$

Fig. 4.9. Calculul coeficienților regresiei liniare simple pe baza unui set de observații

Figura 4.9 arată un exemplu numeric în care sunt calculate valorile necesare pentru determinarea ecuației drepte de regresie. Se folosesc sume precum Σx , Σy , Σx^2 și Σxy , iar pe baza acestora se obțin valorile coeficienților. Rezultatul final este o ecuație de forma $y = a + bx$, care poate fi utilizată pentru estimarea unor valori noi.

În regresia liniară multiplă, modelul are forma generală:

$$\hat{y} = a + b_1 * x_1 + b_1 * x_1 + \dots + b_n * x_n, \quad (5)$$

unde: $x_1, x_2, \dots, x_n$ sunt variabilele independente, iar $b_1, b_2, \dots, b_n$ sunt coeficienții corespunzători acestora.

Fiecare coeficient indică influența predictorului asociat asupra variabilei dependente, presupunând că ceilalți predictorii rămân constanți. Această interpretare este importantă deoarece permite analiza contribuției fiecărei variabile la predicție. Spre deosebire de regresia simplă, unde există o singură pantă, aici există câte un coeficient pentru fiecare variabilă independentă. Modelul poate integra simultan mai multe surse de informație și poate estima o variabilă dependentă pe baza unui ansamblu de factori.

Pentru două variabile independente, modelul poate fi interpretat geometric ca un plan. În acest caz, se caută o suprafață care se poziționează cât mai bine printre punctele observate. În aplicațiile reale, se pot utiliza mult mai mulți predictor, dar principiul rămâne același: se determină acei coeficienți care minimizează diferența dintre valorile reale și cele estimate.

Figura 4.10 ilustrează modul în care o variabilă dependentă poate fi estimată pe baza a două variabile independente. Reprezentarea matricială este utilă deoarece permite exprimarea compactă a regresiei multiple. În loc să scriem manual fiecare ecuație, datele pot fi organizate într-o matrice X , coeficienții într-un vector b , iar predicțiile într-un vector $\hat{y}$.

$$X = \begin{pmatrix} 1 & 1 & 4 \\ 1 & 2 & 5 \\ 1 & 3 & 8 \\ 1 & 4 & 2 \end{pmatrix} \quad y = \begin{pmatrix} 1 \\ 6 \\ 8 \\ 12 \end{pmatrix}$$

x1	x2	Y
Vânzări Produs 1	Vânzări Produs 2	Vânzări Săptămânale
1	4	1
2	5	6
3	8	8
4	2	12

Fig. 4.10. *Predicția unei variabile Y pe baza a două variabile independente*

Forma matricială a regresiei este:

$$\hat{y} = X * b, \tag{6}$$

unde: X este matricea datelor de intrare, iar b este vectorul coeficienților.

Această formulare este importantă pentru implementarea algoritmilor în sisteme informatice, deoarece operațiile matriciale pot fi realizate eficient și pot fi extinse la seturi de date mari.

Figura 4.11 prezintă un exemplu de calcul în care se utilizează matricele pentru estimarea coeficienților. În practică, bibliotecile de învățare automată realizează aceste

calcule automat, însă înțelegerea formei matriciale ajută la interpretarea modului în care modelul transformă datele de intrare în predicții numerice.

$$\begin{aligned}
 X^T X &= \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 4 & 5 & 8 & 2 \end{pmatrix} \begin{pmatrix} 1 & 1 & 4 \\ 1 & 2 & 5 \\ 1 & 3 & 8 \\ 1 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 4 & 10 & 19 \\ 10 & 30 & 46 \\ 19 & 46 & 109 \end{pmatrix} \\
 (X^T X)^{-1} &= \begin{pmatrix} 4 & 10 & 19 \\ 10 & 30 & 46 \\ 19 & 46 & 109 \end{pmatrix}^{-1} = \begin{pmatrix} 3.15 & -0.59 & -0.30 \\ -0.59 & 0.2 & 0.016 \\ -0.3 & 0.016 & 0.054 \end{pmatrix} \\
 (X^T X)^{-1} X^T &= \begin{pmatrix} 3.15 & -0.59 & -0.30 \\ -0.59 & 0.2 & 0.016 \\ -0.3 & 0.016 & 0.054 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 4 & 5 & 8 & 2 \end{pmatrix} = \begin{pmatrix} 0.05 & 0.47 & -1.02 & 0.19 \\ -0.32 & -0.098 & 0.155 & 0.26 \\ -0.065 & 0.05 & 0.185 & -0.125 \end{pmatrix} \\
 = ((X^T X)^{-1} X^T) Y &= \begin{pmatrix} 0.05 & 0.47 & -1.02 & 0.19 \\ -0.32 & -0.098 & 0.155 & 0.26 \\ -0.065 & 0.05 & 0.185 & -0.125 \end{pmatrix} \begin{pmatrix} 1 \\ 6 \\ 8 \\ 12 \end{pmatrix} = \begin{pmatrix} -1.69 \\ 3.48 \\ -0.05 \end{pmatrix} \\
 \mathbf{y} &= -1.69 + 3.48x_1 - 0.05x_2
 \end{aligned}$$

Fig. 4.11. Exemplu de calcul matricial pentru regresia liniară multiplă

Regresia poate include și variabile categoricale, dar acestea trebuie transformate într-o formă numerică. Variabilele ordinale pot fi codificate prin valori care păstrează ordinea, cum ar fi calificativele Excelent, Foarte Bine, Bine, Suficient și Insuficient. Variabilele nominale, care nu au ordine naturală, sunt de obicei transformate prin One-Hot Encoding. Astfel, fiecare categorie devine o coloană binară separată.

	Calculatoare	Biologie	Fizică
Calculatoare	1	0	0
Biologie	0	1	0
Fizică	0	0	1

Fig. 4.12. Tratarea variabilelor categoricale în modele de regresie

Figura 4.12 evidențiază modul de transformare al variabilelor ordinale. În acest caz, atribuirea unor numere arbitrare poate introduce relații false între categorii, motiv pentru care se utilizează codificarea de tip *One-Hot*. Această etapă este esențială pentru ca modelul să poată prelucra corect datele.

4.3. Funcții de cost și evaluarea modelelor de regresie

Pentru a evalua un model de regresie, trebuie măsurată diferența dintre valorile reale și valorile prezise. Această diferență se numește eroare sau reziduu. Dacă pentru un exemplu valoarea reală este y , iar valoarea estimată este $\hat{y}$, atunci eroarea poate fi calculată ca $y - \hat{y}$. Un model bun trebuie să producă erori mici pentru cât mai multe exemple [19].

Figura 4.13 prezintă eroarea ca distanță între valoarea așteptată și valoarea estimată. Punctele reprezintă observațiile reale, iar linia reprezintă modelul. Pentru fiecare punct, se poate calcula diferența verticală față de linie. Aceste diferențe sunt utilizate pentru optimizarea modelului și pentru evaluarea calității predicției.

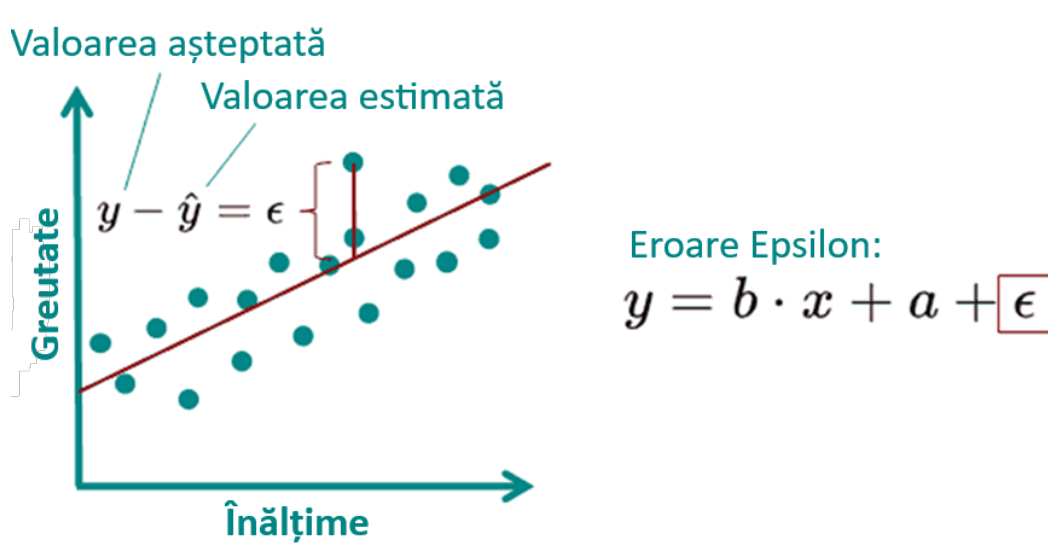


Fig. 4.13. Eroarea de estimare în regresia liniară

O funcție de cost agregă erorile individuale într-o singură valoare numerică. Această valoare permite compararea modelelor și identificarea parametrilor optimi. Dacă funcția de cost este mică, modelul produce predicții apropiate de valorile reale. Dacă funcția de cost este mare, modelul are performanțe slabe sau nu descrie corect relația din date.

Astfel, rolul funcției de cost este de a oferi o măsură numerică a erorii modelului și permite optimizarea parametrilor. În loc să fie analizată separat fiecare eroare, funcția de cost sintetizează comportamentul modelului pe întregul set de date. Din acest motiv, alegerea funcției de cost influențează direct modul în care modelul este antrenat.

O primă măsură intuitivă este eroarea medie, obținută prin media diferențelor dintre valorile reale și valorile prezise. Totuși, această măsură poate fi problematică, deoarece erorile pozitive și negative se pot anula reciproc. Un model poate avea erori mari, dar o eroare medie apropiată de zero, dacă unele predicții sunt prea mari și altele prea mici.

Pentru a evita anularea erorilor, se folosesc măsuri care transformă diferențele în valori pozitive. Eroarea medie pătratică (*Mean Squared Error* - MSE) ridică fiecare eroare la pătrat și calculează media acestor valori. Formula este:

$$MSE = \frac{1}{n} * \sum (y_i - \hat{y}_i)^2 \quad (7)$$

MSE penalizează puternic erorile mari, deoarece diferențele sunt ridicate la pătrat. Acest lucru este util atunci când se dorește ca modelul să evite predicțiile foarte îndepărtate de valorile reale. Pe de altă parte, sensibilitatea ridicată la valori extreme poate fi un dezavantaj atunci când setul de date conține outlieri.

Eroarea medie poate fi înșelătoare atunci când diferențele pozitive și negative se compensează. MSE elimină această problemă prin ridicarea la pătrat a erorilor, dar accentuează influența erorilor mari. Din acest motiv, interpretarea MSE trebuie făcută în contextul datelor analizate.

O altă măsură importantă este eroarea absolută medie (*Mean Absolute Error* - MAE). Aceasta calculează media valorilor absolute ale diferențelor dintre valorile reale și cele prezise:

$$MAE = \frac{1}{n} * \sum |y_i - \hat{y}_i| \quad (8)$$

MAE este ușor de interpretat deoarece se exprimă în aceeași unitate de măsură ca variabila țintă. De exemplu, dacă se estimează un preț, MAE indică abaterea medie în unități monetare. Spre deosebire de MSE, MAE nu penalizează atât de puternic erorile mari, fiind mai robustă în prezența unor valori extreme.

Root Mean Squared Error (RMSE) este rădăcina pătrată a MSE:

$$RMSE = \frac{1}{n} * \sqrt{\sum (y_i - \hat{y}_i)^2} \quad (9)$$

RMSE are avantajul că se exprimă în aceeași unitate ca variabila țintă, dar păstrează sensibilitatea la erori mari specifică MSE. De aceea, RMSE este utilizat frecvent ca indicator principal de performanță pentru modelele de regresie, mai ales atunci când erorile mari sunt considerate mai grave decât erorile mici.

Un alt indicator important este coeficientul de determinare, notat R^2 . Acesta măsoară proporția variației variabilei dependente care este explicată de variabilele independente incluse în model. Formula generală poate fi exprimată astfel:

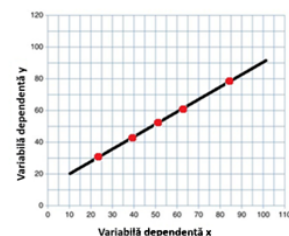
$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}, \quad (10)$$

unde: SS_{res} este suma pătratelor reziduurilor, iar SS_{tot} este suma totală a pătratelor față de media valorilor reale.

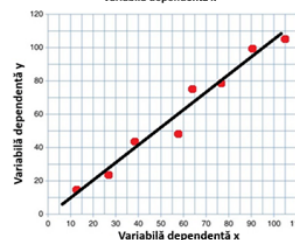
Valoarea R^2 este utilizată pentru a interpreta cât de bine explică modelul variația datelor. O valoare apropiată de 1 indică faptul că modelul explică o mare parte din variație, iar o valoare apropiată de 0 indică o capacitate explicativă redusă.

Figura 4.14 ilustrează trei situații. Pentru $R^2 = 1$, modelul explică întreaga variație a valorilor y . Pentru $R^2 = 0,83$, aproximativ 83% din variația variabilei dependente este explicată de model. Pentru $R^2 = 0$, variabilele independente nu explică variația lui y . În practică, valoarea R^2 trebuie interpretată împreună cu erorile MAE, MSE sau RMSE și cu analiza grafică a reziduurilor.

$R^2 = 1$ Toată variația valorilor y este explicată de valorile x



$R^2 = 0.83$ 83 % din variația valorilor y este explicată de valorile x



$R^2 = 0$ Variația valorilor y nu este influențată deloc de valorile x

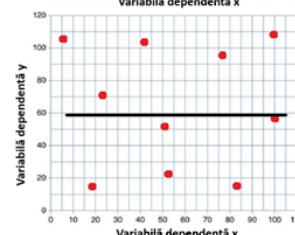


Fig. 4.14. Exemple de interpretare pentru valori diferite ale coeficientului R^2

4.4. Regresia Logistică

Deși regresia liniară este destinată estimării unei variabile continue, apare uneori tentația de a fi utilizată și pentru clasificare binară. În acest caz, etichetele pot fi codificate numeric, de exemplu 0 și 1 sau -1 și 1, iar modelul liniar poate produce o valoare care este apoi transformată într-o decizie de clasă. Totuși, această abordare are limitări importante, deoarece modelul liniar nu este optimizat pentru clasificare și poate produce valori în afara intervalului admis pentru probabilități.

Astfel, în cazul în care clasele sunt codificate numeric, modelul poate genera o valoare apropiată de o clasă sau de alta. Cu toate acestea, valorile prezise de regresia liniară nu sunt constrânse între 0 și 1, ceea ce face dificilă interpretarea lor ca probabilități. Din acest motiv, pentru astfel de probleme se utilizează regresia logistică.

Regresia logistică este utilizată pentru probleme în care variabila țintă este binară. Modelul estimează probabilitatea ca o instanță să aparțină clasei pozitive. Dacă

probabilitatea este mai mare decât un prag ales (de obicei 0,5), instanța este clasificată în clasa 1; altfel este clasificată în clasa 0. Astfel, regresia logistică transformă o problemă de estimare într-o decizie de clasificare.

Problema utilizării unei linii de regresie pentru o variabilă binară este că linia poate produce valori mai mici decât 0 sau mai mari decât 1. În plus, relația dintre predictor și probabilitatea evenimentului nu este de obicei liniară pe întregul domeniu. Pentru valori extreme ale predictorului, probabilitatea trebuie să se apropie de 0 sau de 1, fără să depășească aceste limite.

Figura 4.15 arată de ce regresia liniară nu este potrivită pentru o variabilă dependentă binară. Modelul liniar poate produce predicții neobservabile, adică valori care nu pot reprezenta o probabilitate validă. Pentru a evita această problemă, se aplică o transformare care restrânge rezultatul în intervalul $[0, 1]$.

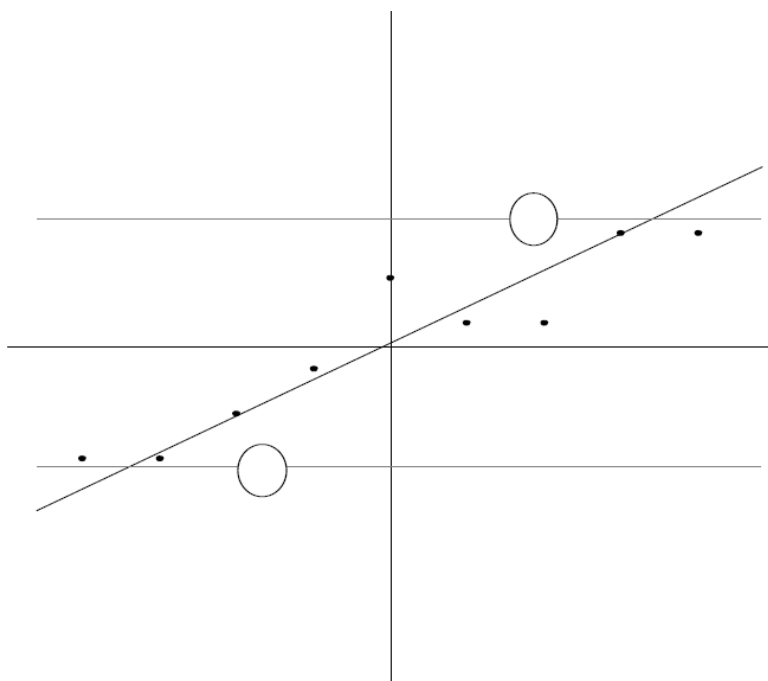


Fig. 4.15. Limitări ale regresiei liniare atunci când variabila dependentă este binară

Regresia logistică pornește de la o combinație liniară a predictorilor, similar cu regresia liniară [15]:

$z = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$	(11)
---	--------

Această valoare z poate lua orice valoare reală, de la minus infinit la plus infinit. Pentru a o transforma într-o probabilitate, se aplică funcția logistică, cunoscută și sub numele de funcție sigmoidă:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (12)$$

Funcția logistică transformă orice valoare reală într-o valoare cuprinsă între 0 și 1. Această proprietate o face potrivită pentru modelarea probabilității unui eveniment. În regresia logistică, rezultatul modelului este interpretat ca probabilitatea condiționată $P(y = 1 | x)$, adică probabilitatea ca instanța descrisă de caracteristicile x să aparțină clasei pozitive [2].

Figura 4.16 prezintă forma caracteristică a funcției logistice. Pentru valori foarte mici ale lui z , funcția tinde către 0. Pentru valori foarte mari, funcția tinde către 1. În zona centrală, funcția are o creștere rapidă, iar valoarea 0,5 apare atunci când z este egal cu 0. Această formă este potrivită pentru decizii binare, deoarece permite interpretarea rezultatelor ca probabilități.

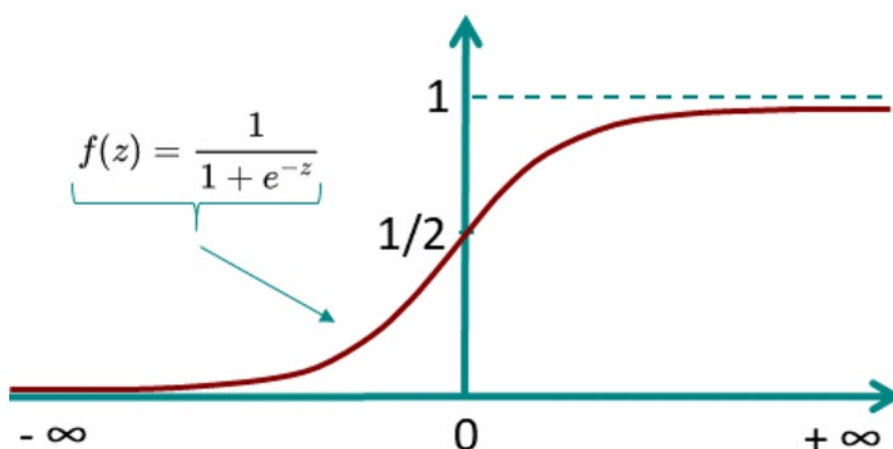


Fig. 4.16. Funcția logistică și intervalul valorilor sale

Figura 4.17 arată ecuația funcției logistice și modul în care aceasta transformă combinația liniară a predictorilor. Dacă probabilitatea estimată este apropiată de 1, modelul indică apartenența la clasa pozitivă. Dacă probabilitatea este mai apropiată de 0, modelul indică apartenența la clasa negativă. Pragul de decizie poate fi ajustat în funcție de aplicație și de costul erorilor.

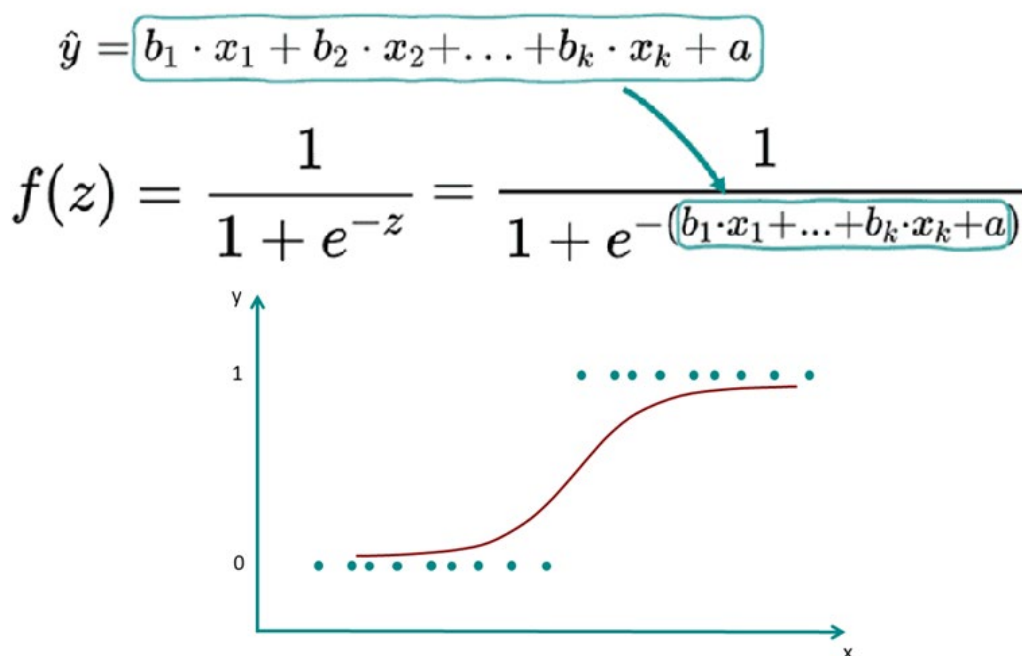


Fig. 4.17. Ecuația funcției logistice și reprezentarea sa grafică

Regresia logistică are o abordare probabilistică. Modelul nu returnează doar o etichetă, ci estimează probabilitatea apartenenței la o clasă. Această probabilitate poate fi folosită pentru decizii mai flexibile. De exemplu, într-un sistem de detecție, se poate alege un prag mai mic pentru a reduce riscul de a rata cazurile pozitive, chiar dacă acest lucru crește numărul de alarme false.

Figura 4.18 evidențiază faptul că regresia logistică învață direct distribuția de probabilitate $P(Y|X)$. Pentru fiecare instanță, modelul estimează probabilitatea clasei pozitive în funcție de valorile predictorilor. Această abordare este foarte utilă în aplicațiile

în care nu este suficientă o decizie rigidă, ci este necesară și o măsură a încrederii în predicție.

$$P(y = 1|x_1, x_2, \dots, x_n) = \frac{1}{1 + e^{-(b_1x_1 + b_2x_2 + \dots + b_nx_n + a)}}$$

$$\begin{aligned} P(y = 0|x_1, x_2, \dots, x_n) &= 1 - P(y = 1) \\ &= 1 - \frac{1}{1 + e^{-(b_1x_1 + b_2x_2 + \dots + b_nx_n + a)}} \end{aligned}$$

Fig. 4.18. Abordarea probabilistică în regresia logistică

Un exemplu intuitiv este predicția promovării unui examen în funcție de numărul de ore de studiu. Variabila independentă este numărul de ore, iar variabila dependentă este binară: 1 pentru promovat și 0 pentru nepromovat. Modelul logistic poate estima probabilitatea de promovare pentru un anumit număr de ore studiate. Pe măsură ce numărul de ore crește, probabilitatea estimată crește, dar rămâne întotdeauna între 0 și 1.

Figura 4.19 prezintă un exemplu în care datele arată dacă un student a promovat sau nu în funcție de timpul de studiu. Pentru o valoare x dată, se calculează mai întâi combinația liniară, apoi se aplică funcția logistică. Rezultatul este o probabilitate. De exemplu, pentru un număr mic de ore, probabilitatea poate fi sub pragul de decizie, iar modelul va prezice nepromovarea. Pentru un număr mai mare de ore, probabilitatea poate depăși pragul, iar modelul va prezice promovarea.

Regresia logistică poate fi extinsă și pentru probleme cu mai multe clase, numită regresie logistică multinomială. În loc să existe doar două rezultate posibile, modelul estimează probabilități pentru mai multe clase și alege clasa cu probabilitatea cea mai mare. În cazul regresiei logistice, deoarece modelul este utilizat pentru clasificare, se aplică indicatorii specifici clasificării (acuratețe, precizie, recall și scor F1), ce au fost prezentate în capitolul anterior.

Ore de studiu (x_i)	0.5	0.75	1.00	1.25	1.50	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Promovat (y_i)	0	0	0	0	0	0	0	1	0	1	0	1	0	1	1	1	1	1	1

Ecuatie de regresie: $\hat{y}_i = a + b_1 X_i$ Determinăm valorile pentru:
 $a = -4.1$
 $b_1 = 1.5$

De exemplu, pentru $x = 2$: $\hat{y}_i = a + b_1 * 2 = -4.1 + 1.5 * 2 = -1.1$

Aplicăm funcția logistică: $P(y = 1|x) = \frac{1}{1 + e^{-\hat{y}}} = \frac{1}{1 + e^{1.1}} \approx 0.25 \rightarrow \text{clasa 0}$

Similar pentru $x = 4$: $\hat{y}_i = a + b_1 * 4 = -4.1 + 1.5 * 4 = 1.9$

$$P(y = 1|x) = \frac{1}{1 + e^{-\hat{y}}} = \frac{1}{1 + e^{-1.9}} \approx 0.87 \rightarrow \text{clasa 1}$$

Fig. 4.19. Exemplu de regresie logistică: probabilitatea de promovare în funcție de orele de studiu

Astfel, regresia liniară și regresia logistică reprezintă două metode fundamentale în învățarea supervizată. Regresia liniară este potrivită pentru predicția valorilor continue, iar regresia logistică este potrivită pentru estimarea probabilităților și clasificarea binară. În ambele cazuri, performanța modelului depinde de calitatea datelor, de alegerea corectă a variabilelor, de tratarea valorilor extreme și de evaluarea pe date care nu au fost folosite la antrenare [16].

5. Capitolul 5. Tehnici de învățare nesupervizată: Clustering

Învățarea nesupervizată urmărește identificarea structurilor ascunse într-un set de date pentru care nu sunt disponibile etichete de clasă. Spre deosebire de clasificare și regresie, algoritmul nu primește o variabilă țintă, ci trebuie să descopere singur grupări, regularități și relații între observații. Clustering-ul reprezintă una dintre cele mai importante tehnici din această categorie și are ca obiectiv organizarea obiectelor în grupuri omogene, denumite clustere.

Capitolul prezintă conceptele fundamentale ale grupării datelor, măsurile de proximitate, clustering-ul ierarhic, algoritmul K-Means și algoritmul DBSCAN. Sunt analizate atât mecanismele de funcționare, cât și criteriile de evaluare, avantajele, limitările și situațiile în care fiecare metodă este adecvată.

5.1. Introducere în învățarea nesupervizată

Paradigmele de învățare automată se diferențiază în primul rând prin tipul informației furnizate algoritmului. În învățarea supervizată, fiecare exemplu este însoțit de o etichetă, iar modelul învață relația dintre intrări și ieșiri. În învățarea nesupervizată sunt disponibile numai observațiile, fără răspunsuri corecte cunoscute. Algoritmul trebuie să descopere singur structura datelor, ceea ce transformă analiza într-un proces exploratoriu.

Lipsa unei variabile țintă înseamnă că nu putem formula direct o problemă clasică de clasificare sau regresie. În schimb, se urmărește identificarea unor șabloane precum grupuri compacte de observații, relații de proximitate, structuri ierarhice sau reprezentări de dimensiune redusă. Rezultatul nu este neapărat unic, deoarece aceeași colecție de date poate fi grupată diferit în funcție de obiectiv, măsura de similaritate și algoritmul utilizat (Figura 5.1).

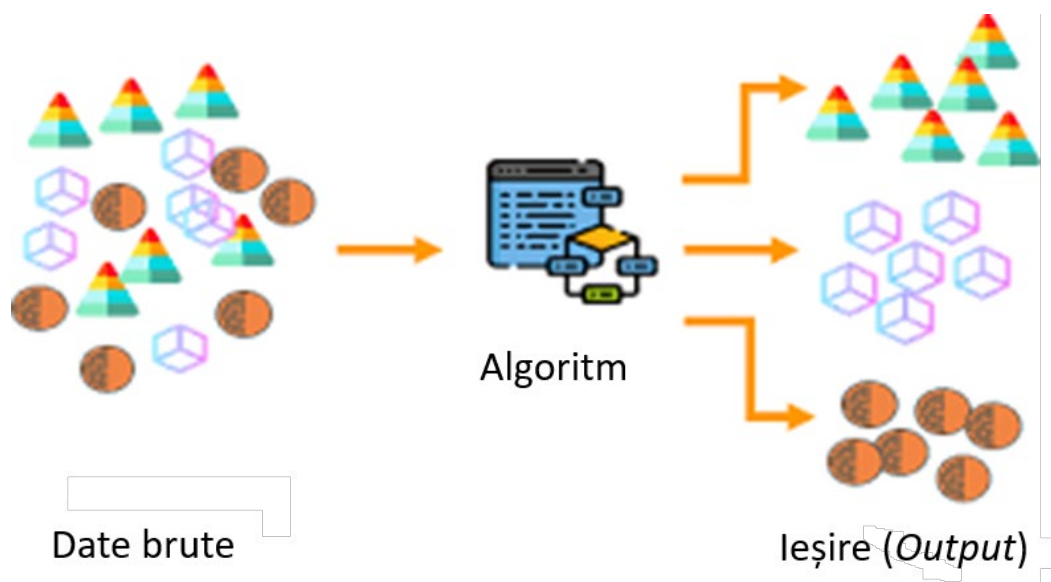


Fig. 5.1. Transformarea datelor brute în grupuri fără utilizarea etichetelor

În practică, învățarea nesupervizată este utilizată frecvent ca etapă preliminară pentru înțelegerea unui set de date. Grupările obținute pot sugera categorii naturale, pot evidenția observații neobișnuite și pot ghida selectarea caracteristicilor sau construirea ulterioară a unor modele de învățare supervizată.

5.2. Conceptul de clustering și măsuri de proximitate

Clustering-ul este procesul de împărțire a unui set de obiecte în grupuri astfel încât obiectele din același grup să fie cât mai asemănătoare, iar obiectele din grupuri diferite să fie cât mai diferite, așa cum ilustrează Figura 5.2. Un cluster nu este definit în prealabil, ci este identificat pe baza relațiilor de similaritate existente între date. Din acest motiv, clustering-ul nu trebuie confundat cu o segmentare arbitrară, cu rezultatul unei interogări sau cu o clasificare bazată pe etichete cunoscute.

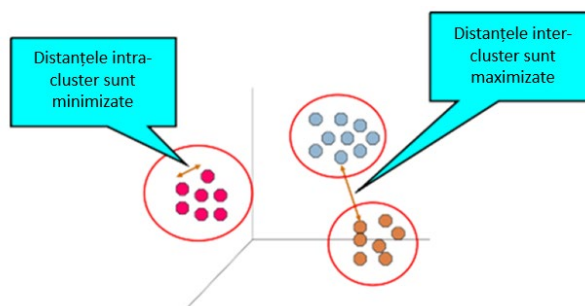


Fig. 5.2. Principiul general al formării clusterelor

În majoritatea cazurilor, un cluster este reprezentat printr-un centroid, adică un punct care sintetizează poziția membrilor săi. Figura 5.3 ilustrează modul în care centroidul este poziționat în centrul clusterului și servește drept punct de referință pentru gruparea instanțelor. Pentru date numerice, centroidul este de regulă media elementelor din cluster. Această reprezentare este eficientă pentru grupuri compacte, dar poate deveni neadecvată pentru forme complexe sau densități foarte diferite.

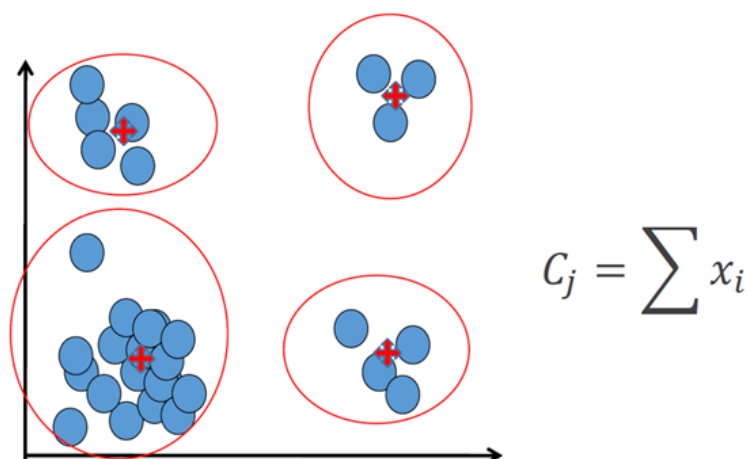


Fig. 5.3. Centroidul și limita unui cluster

Calitatea unei grupări se bazează pe două obiective complementare: minimizarea distanțelor intracluster și maximizarea distanțelor dintre cluster. Aceste două criterii exprimă coeziunea internă și separarea externă. O soluție bună trebuie să mențină

observațiile similare împreună, fără a combina în același grup regiuni distincte ale spațiului de date.

Similaritatea este o măsură numerică a asemănării dintre două obiecte și are valori mai mari atunci când obiectele sunt mai apropiate din punct de vedere al caracteristicilor. Disimilaritatea exprimă diferența dintre obiecte și are valori mici pentru observații asemănătoare. Termenul de proximitate este utilizat generic pentru ambele tipuri de măsuri.

Pentru date numerice se utilizează adesea distanța euclidiană. Pentru două puncte x și y cu r caracteristici, distanța este rădăcina pătrată a sumei diferențelor pătrate dintre componente. Alegerea măsurii depinde însă de tipul datelor. Distanța Manhattan poate fi mai robustă în anumite situații, iar similaritatea cosinus este frecvent utilizată pentru documente și vectori rari. Pentru atribute categoriale sunt necesare măsuri adaptate, bazate pe potrivirea valorilor sau pe ponderi definite în funcție de domeniu.

Scalarea caracteristicilor este esențială înaintea aplicării algoritmilor bazați pe distanță. Dacă o caracteristică are valori de ordinul miilor, iar alta valori între zero și unu, prima va domina calculul proximității. Normalizarea sau standardizarea permite ca fiecare atribut să contribuie în mod proporțional la calculul distanțelor, evitând dominarea rezultatului de către variabilele cu valori numerice mari.

Algoritmii de grupare pot fi utilizați pentru clasificarea persoanelor în categorii de dimensiuni, astfel încât să fie stabilite mărimi potrivite pentru produse precum tricourile, evitând atât personalizarea costisitoare, cât și soluția universală de tip „*one-size-fits-all*”. În marketing, clusteringul permite segmentarea clienților în funcție de asemănările dintre aceștia, facilitând realizarea unor campanii țintite. De asemenea, tehnica poate fi aplicată colecțiilor de documente, care pot fi organizate automat în grupuri tematice pe baza similitudinii conținutului.

5.3. Clustering Ierarhic

Clustering-ul ierarhic produce un ansamblu de clustere imbricate, organizate sub forma unui arbore. Structura rezultată poate fi vizualizată printr-o dendrogramă, care înregistrează ordinea în care clusterelor sunt fuzionate sau divizate. Avantajul principal este că nu este necesară alegerea inițială a unui număr fix de clustere; acesta poate fi

stabilit ulterior prin tăierea dendrogramei la un anumit nivel. Modul de reprezentare a unui cluster ierarhic este ilustrat în Figura 5.4.

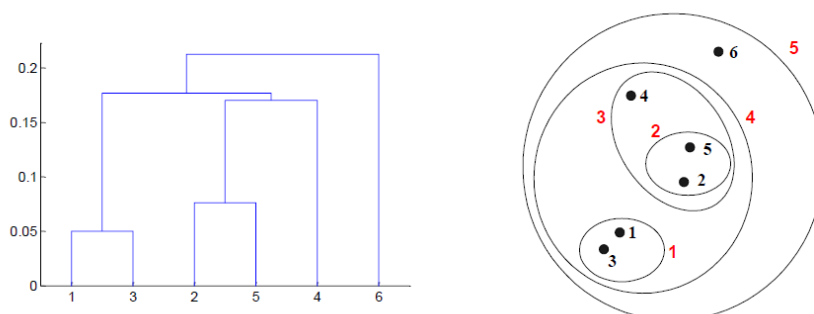


Fig. 5.4. Reprezentarea ierarhică a clusterelor prin dendrogramă

Există două strategii principale de clustering ierarhic. Metoda aglomerativă pornește cu câte un cluster pentru fiecare observație și unește succesiv cele mai apropiate cluster. Metoda divizivă pornește de la un singur cluster care conține toate observațiile și îl împarte treptat. În practică, varianta aglomerativă este utilizată mai frecvent datorită simplității sale conceptuale și disponibilității unor criterii clare de legătură.

Algoritmul aglomerativ calculează mai întâi matricea de proximitate, consideră fiecare punct drept un cluster și repetă două operații: unește cele mai apropiate două cluster și actualizează matricea de distanțe. Procesul continuă până când rămâne un singur cluster sau până când este atins numărul dorit de grupuri.

Pași metodei de clustering ierarhic aglomerativ sunt următorii:

1. Se calculează matricea de proximitate.
2. Fiecare punct de date va reprezenta un cluster.
3. Se repetă până când rămâne un singur cluster.
 - Se unesc cele mai apropiate două cluster.
 - Se actualizează matricei de proximitate

Figura 5.5 ilustrează etapa inițială a algoritmului de clustering ierarhic aglomerativ. În partea stângă sunt reprezentate observațiile din setul de date, fiecare punct constituind inițial un cluster individual. Pentru a determina ce cluster vor fi reunite la

fiecare pas al algoritmului, se construiește matricea de proximitate, care conține distanțele dintre toate perechile de observații. Pe baza valorilor din această matrice sunt identificate, la fiecare iterație, cele mai apropiate două clustere, care sunt apoi reunite într-un singur cluster.

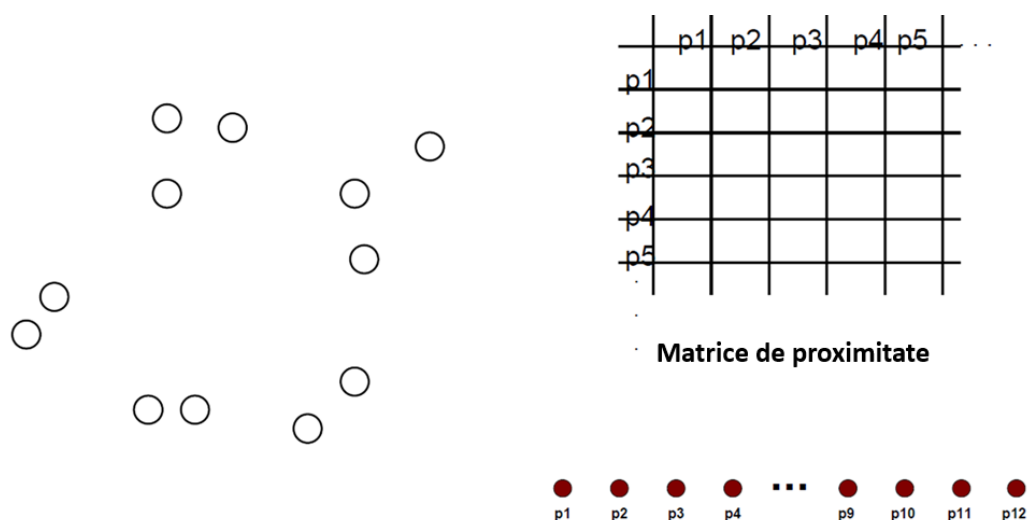


Fig. 5.5. Matricea de proximitate utilizată în clusteringul ierarhic aglomerativ

Figura 5.6 prezintă etapa următoare a algoritmului de clustering ierarhic aglomerativ, după realizarea primei fuziuni. În exemplul ilustrat, clusterelor C2 și C5 au fost reunite, rezultând noul cluster $C2 \cup C5$. În consecință, matricea de proximitate este actualizată pentru a include distanțele dintre noul cluster și celelalte clustere existente, iar dendrograma evidențiază prima etapă a construirii ierarhiei.

După formarea unui nou cluster, procesul de grupare continuă prin identificarea următoarelor clustere care urmează să fie reunite. Pentru aceasta, este necesară definirea unei metode de calcul a distanței dintre clustere, deoarece acestea pot conține mai multe observații. În acest scop sunt utilizate diferite criterii de legătură, precum legătură simplă (single linkage), legătură complete (complete linkage), legătură medie (average linkage) sau distanța dintre centroizi. Alegerea criteriului de legătură influențează modul de construire a ierarhiei și structura finală a clusterelor obținute.

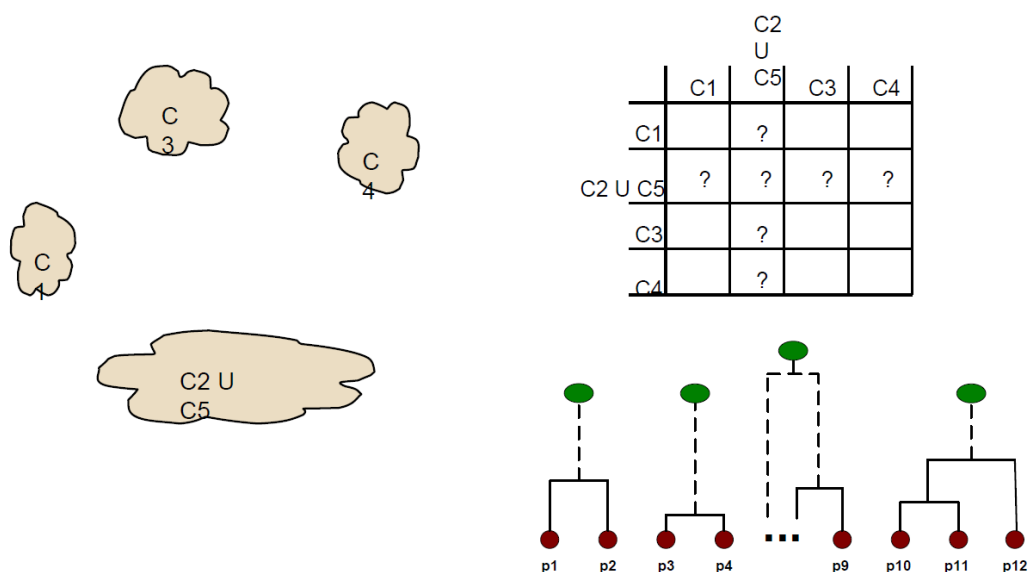


Fig. 5.6. Actualizarea clusterelor și a matricei de proximitate

În legătura simplă, distanța este cea mai mică distanță dintre două puncte aparținând clusterelor diferite.

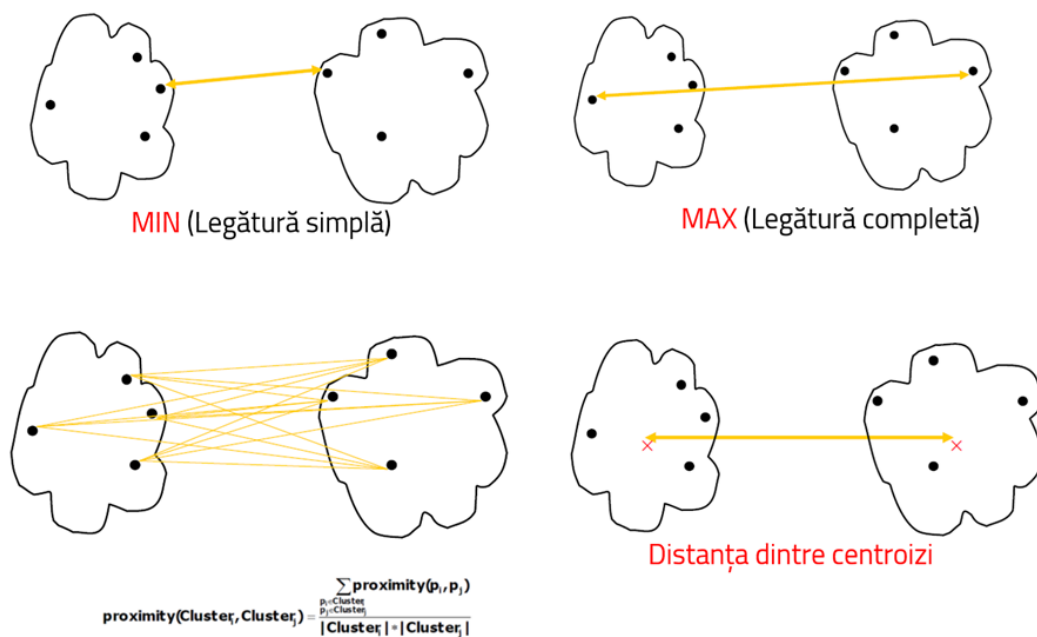


Fig. 5.7. Modalități de calcul al distanței dintre cluster

În legătura completă se utilizează cea mai mare distanță dintre puncte, ceea ce favorizează clustere compacte. Legătura medie calculează media tuturor distanțelor perechi și oferă un compromis între cele două extreme. Figura 5.7 ilustrează modalitatea de determinare a distanței dintre 2 clustere prin fiecare metodă.

Distanța dintre centroizi utilizează centrele geometrice ale clusterelor. Alegerea criteriului de legătură poate modifica semnificativ dendrograma și rezultatul final, așa cum arată și Figura 5.8. Din acest motiv, metoda trebuie selectată în funcție de forma așteptată a grupurilor și de sensibilitatea acceptată la zgomot sau valori aberante.

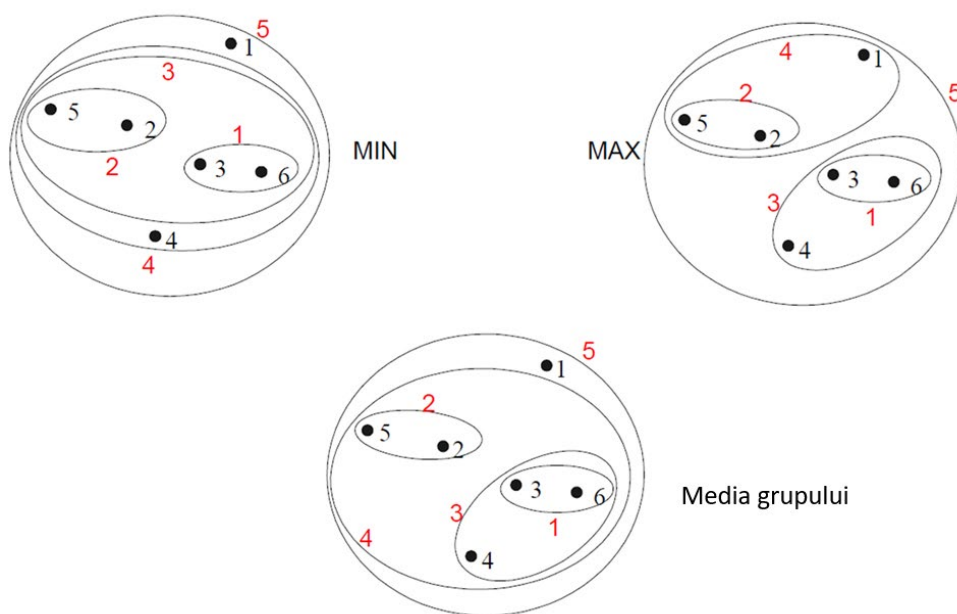


Fig. 5.8. Compararea rezultatelor pentru diferite criterii de legătură

Clustering-ul ierarhic este potrivit pentru seturi de date mici și medii, pentru situații în care se dorește o taxonomie sau pentru analiza relațiilor la mai multe niveluri de granularitate. Costul său de calcul și memorie este însă mai mare decât al metodelor partiționale, ceea ce limitează aplicarea directă pe volume foarte mari de date.

5.4. Algoritmul K-Means

K-Means este un algoritm partițional care împarte un set de n observații în k clustere fără suprapunere. Fiecare cluster este reprezentat printr-un centroid, iar fiecare observație este atribuită clusterului al cărui centroid se află la cea mai mică distanță de aceasta. Numărul k este stabilit înainte de rularea algoritmului, ceea ce reprezintă atât un avantaj de control, cât și o limitare atunci când structura datelor nu este cunoscută.

Algoritmul alternează între două etape. În etapa de atribuire, fiecare punct este asociat celui mai apropiat centroid. În etapa de actualizare, centroidul fiecărui cluster este recalculat ca media punctelor atribuite. Etapele se repetă până când centroizii nu se mai modifică semnificativ sau când punctele nu mai schimbă clusterul.

Algoritmul K-Means este un proces iterativ care urmărește împărțirea unui set de date în k clustere, astfel încât observațiile din același cluster să fie cât mai asemănătoare, iar cele din clustere diferite să fie cât mai distincte. Funcționarea algoritmului poate fi descrisă prin următorii pași:

1. Inițializarea centroidelor. În prima etapă se alege numărul de clustere, k , iar apoi se inițializează aleatoriu k centroizi, notați cu $c_1, c_2, \dots, c_k$. Acești centroizi reprezintă punctele de referință inițiale ale clusterelor. Deoarece poziția lor este aleasă aleatoriu, rezultate diferite pot fi obținute la execuții diferite ale algoritmului.
2. Atribuirea observațiilor clusterelor. După stabilirea centroidelor inițiale, fiecare observație din setul de date este comparată cu toți centroizii existenți. Pentru fiecare punct p , se calculează distanța față de fiecare centroid (de regulă utilizând distanța euclidiană), iar observația este atribuită clusterului al cărui centroid se află la cea mai mică distanță. La finalul acestui pas, toate observațiile sunt repartizate într-unul dintre cele k clustere.
3. Recalcularea centroidelor. După atribuirea tuturor observațiilor, poziția fiecărui centroid este actualizată. Noul centroid al unui cluster se calculează ca media aritmetică a coordonatelor tuturor observațiilor care aparțin aceluși cluster. Astfel, centroidul își modifică poziția astfel încât să reprezinte cât mai bine distribuția actuală a punctelor din cluster.
4. Verificarea criteriului de oprire. După recalcularea centroidelor, se verifică dacă poziția acestora s-a modificat față de iterația anterioară. Dacă unul sau mai mulți

centroizi și-au schimbat poziția, algoritmul revine la Pasul 2, iar observațiile sunt redistribuite în funcție de noile poziții ale centroidelor. Procesul continuă iterativ până când centroizii nu se mai modifică sau modificările sunt mai mici decât un prag prestabilit. În acest moment se consideră că algoritmul a convergit, iar clusterurile obținute reprezintă soluția finală.

Figura 5.9 ilustrează modul de funcționare al algoritmului K-Means. După alegerea numărului de clusteruri ($k = 3$) și inițializarea centroidelor, observațiile sunt atribuite celui mai apropiat centroid, iar poziția acestora este actualizată iterativ până la obținerea configurației finale a clusterurilor.

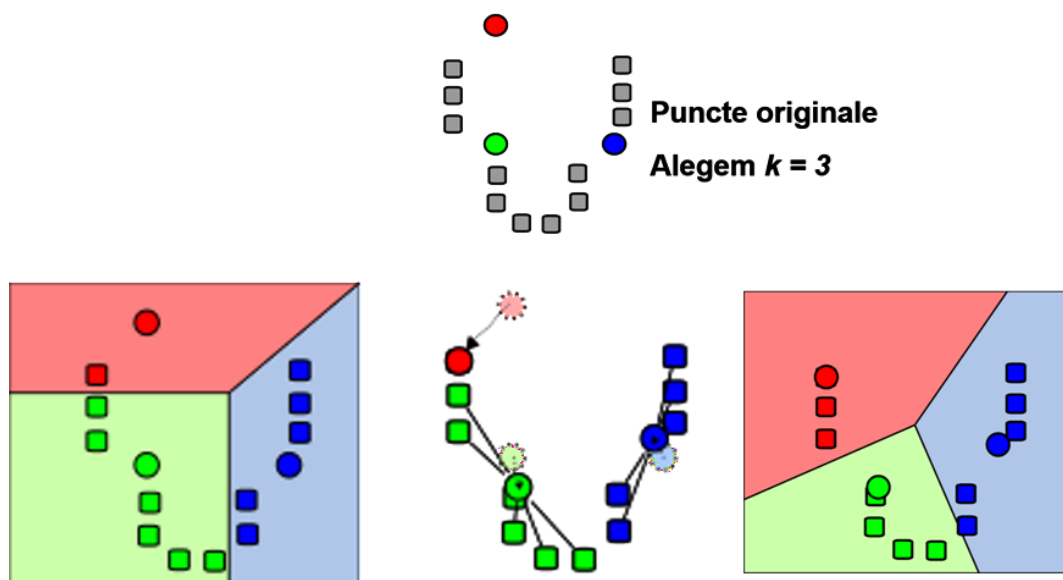


Fig. 5.9. Succesiunea etapelor K-Means pentru $k = 3$

Inițializarea centroizilor influențează rezultatul. Alegerea aleatorie poate conduce la soluții diferite între rulări și la optime locale. O practică uzuală este efectuarea mai multor inițializări și păstrarea soluției cu cea mai mică eroare. Metoda K-Means++ îmbunătățește selecția centrelor inițiale prin distribuirea lor mai uniformă în spațiul datelor.

5.4.1. Funcția obiectiv și criteriul de oprire

K-Means minimizează suma erorilor pătratică, denumită SSE sau *inertia*. Pentru fiecare punct se calculează distanța pătratică față de centroidul clusterului său, iar valorile sunt însumate. O valoare mai mică indică o coeziune internă mai bună. Totuși, SSE scade întotdeauna când numărul de clustere crește, astfel încât nu poate fi utilizată singură pentru alegerea lui k .

Oprirea poate fi stabilită atunci când nicio observație nu mai este realocată, când modificarea centroizilor scade sub un anumit prag sau când reducerea SSE devine neglijabilă. În implementările practice este utilizat și un număr maxim de iterații pentru a controla timpul de execuție.

Complexitatea algoritmului este favorabilă pentru seturi mari de date atunci când k și numărul de iterații sunt moderate. Din acest motiv, K-Means este una dintre cele mai populare metode de clustering. Performanța sa depinde însă de reprezentarea datelor, de scalare și de existența unor clustere aproximativ compacte și de dimensiuni comparabile.

5.4.2. Alegerea numărului de clustere

Metoda Elbow analizează valoarea inertia pentru mai multe valori ale lui k . Curba scade rapid la început, apoi îmbunătățirea devine mai lentă. Punctul în care curba începe să se aplatizeze este considerat un compromis optim între numărul de clustere și calitatea grupării. Metoda este intuitivă, dar, în multe situații, cotul poate fi slab conturat sau ambiguu. Figura 5.10 ilustrează metoda Elbow pentru determinarea numărului optim de clustere. Se observă că eroarea scade odată cu creșterea valorii lui k , iar punctul în care curba începe să se aplatizeze ($k=4$) indică valoarea optimă a lui k , dincolo de care îmbunătățirile devin reduse.

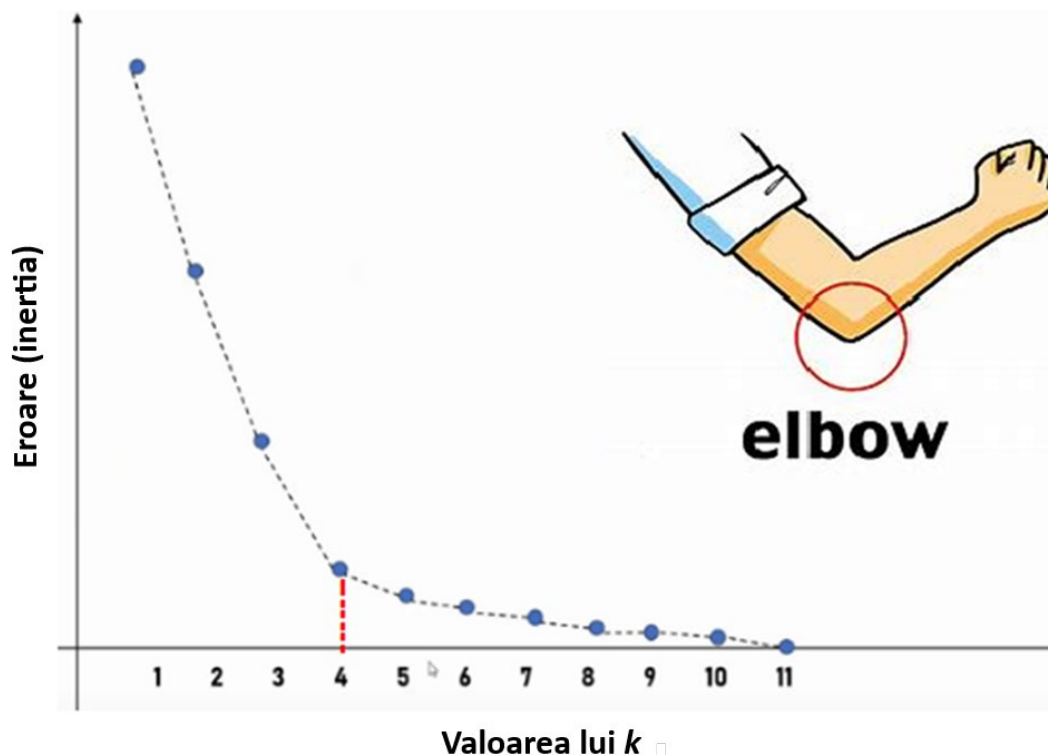


Fig. 5.10. Situație în care alegerea cotului este ambiguă

Coeficientul Silhouette evaluează simultan coeziunea și separarea. Pentru un punct i , $a(i)$ este distanța medie față de punctele din propriul cluster, iar $b(i)$ este distanța medie față de punctele din cel mai apropiat alt cluster. Valoarea $s(i)$ se află între -1 și 1. Valorile apropiate de 1 indică o atribuire bună, valorile apropiate de 0 indică puncte de frontieră, iar valorile negative sugerează o posibilă atribuire greșită. Formula de calcul a coeficientului Silhouette pentru un punct este următoarea:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \quad (13)$$

5.4.3. Exemplu de funcționare

Exemplul din figurile următoare pornește de la un set bidimensional, iar diagrama *scatter plot* pe axa x-y este ilustrată în Figura 5.11.

Se consideră că numărul de cluster este $k = 2$. În prima etapă a algoritmului, este necesară inițializarea centroidelor. Astfel, se aleg aleatoriu două puncte, care vor reprezenta centrozii inițiali ai celor două cluster. Aceste puncte nu trebuie neapărat să coincidă cu observații existente în setul de date, ci pot fi selectate în orice poziție din spațiul atributelor. Figura 5.12 prezintă distribuția inițială a observațiilor și poziția celor doi centroizi aleși, care vor servi drept puncte de referință pentru atribuirea observațiilor în etapele următoare ale algoritmului.

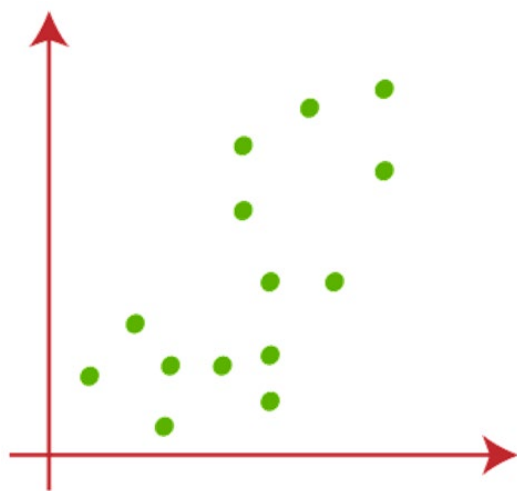


Fig. 5.11. Diagrama *scatter plot* a setului de date

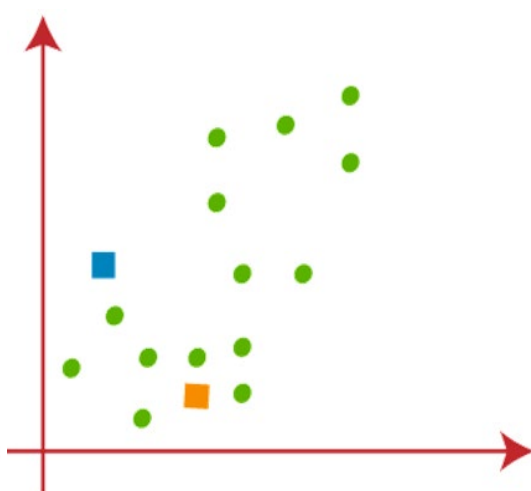


Fig. 5.12. Determinarea centroizilor inițiali

În continuare, fiecare observație este atribuită clusterului al cărui centroid se află la cea mai mică distanță. În acest scop, se calculează distanța dintre fiecare punct din setul de date și cei doi centroizi inițiali, iar observația este asociată celui mai apropiat centroid. Limita dintre cele două cluster este reprezentată de mediatoarea segmentului care unește cei doi centroizi, aceasta delimitând regiunile în care observațiile sunt atribuite fiecărui cluster. Figura 5.13 ilustrează rezultatul acestui prim proces de atribuire a observațiilor.

După prima etapă de atribuire, se observă că unele observații și-au schimbat apartenența la cluster. În consecință, este necesară recalcularea centroidelor, deoarece pozițiile inițiale nu mai reprezintă în mod corespunzător distribuția actuală a observațiilor din fiecare cluster. Noul centroid al fiecărui cluster se determină ca media aritmetică a coordonatelor tuturor observațiilor care îi aparțin. Figura 5.14 ilustrează această etapă de recalculare a centroidelor și continuarea procesului iterativ al algoritmului K-Means.

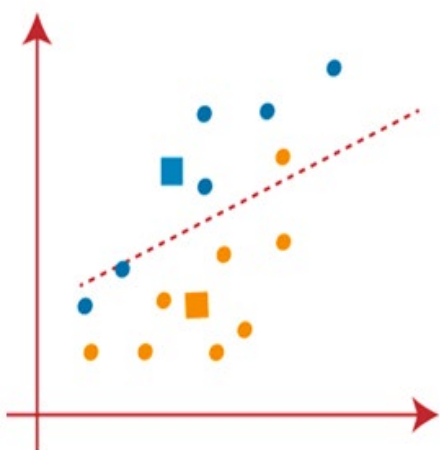


Fig. 5.13. Atribuirea punctelor în cluster

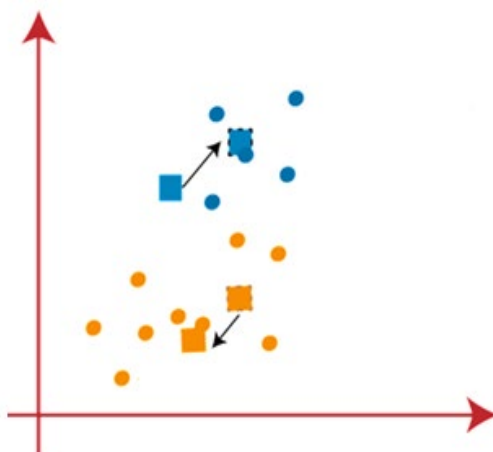


Fig. 5.14. Determinarea noilor centroizi

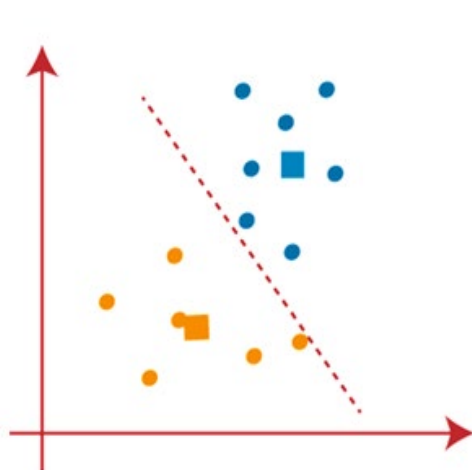


Fig. 5.15. Atingerea criteriului de oprire

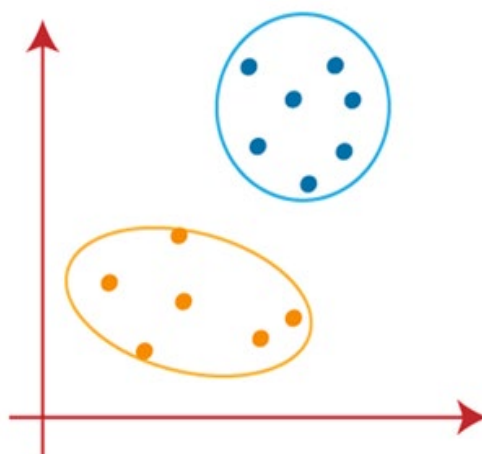


Fig. 5.16. Forma finală a clusterelor

După actualizarea repetată a centroidelor și reatribuirea observațiilor, algoritmul ajunge într-o stare de convergență, în care apartenența observațiilor la cluster nu se

mai modifică (Figura 5.15). În această etapă, poziția centroidelor rămâne stabilă, iar clusterelor obținute reprezintă soluția finală furnizată de algoritmul K-Means. Figura 5.16 ilustrează configurația finală a celor 2 cluster rezultate.

5.4.4. Avantaje și limitări

K-Means este simplu, rapid și ușor de interpretat. Este potrivit pentru seturi mari de date numerice și pentru clusterare aproximativ sferice, bine separate și cu densități similare. Algoritmul oferă o segmentare exclusivă: fiecare observație aparține exact unui cluster.

Limitările apar atunci când grupurile au dimensiuni sau densități foarte diferite, forme non-convexe ori conțin valori aberante, așa cum arată Figura 5.17. Deoarece centroidul este o medie, outlierii îl pot influența semnificativ. Algoritmul necesită alegerea numărului de cluster k , iar poziția inițială a centroidelor poate influența rezultatul final. De asemenea, media nu este definită natural pentru date categoricale, situație în care sunt necesare variante precum K-Modes.

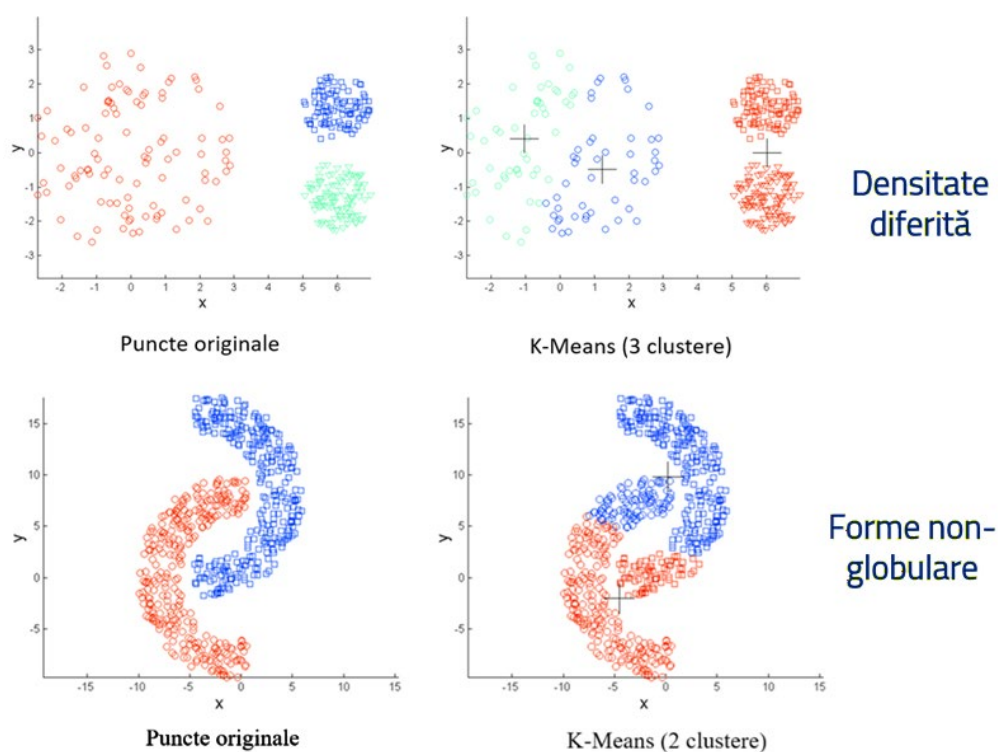


Fig. 5.17. Limitări pentru densități diferite și forme non-globulare

5.5. Algoritmul DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise), definește clusterelor ca regiuni dense separate de regiuni cu densitate redusă. Spre deosebire de K-Means, nu necesită specificarea numărului de clusterelor și poate identifica grupuri de formă arbitrară. În plus, tratează explicit punctele izolate ca zgomot.

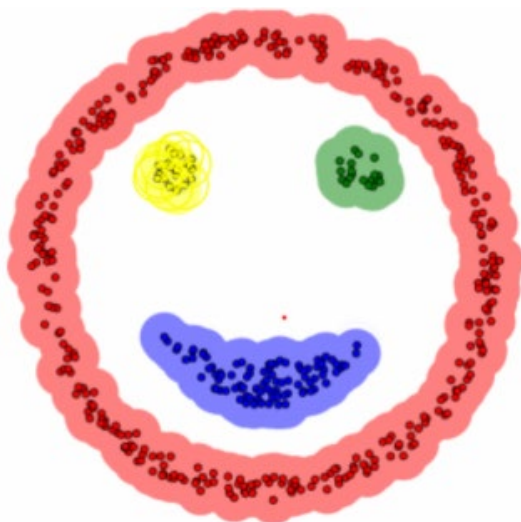


Fig. 5.18. *Principiul clustering-ului bazat pe densitate*

Algoritmul utilizează doi parametri: ϵ , raza vecinătății, și MinPts, numărul minim de puncte necesar pentru ca o regiune să fie considerată densă. Un punct core are cel puțin MinPts observații în vecinătatea sa ϵ . Un punct border se află în vecinătatea unui punct core, dar nu îndeplinește singur condiția de densitate, în timp ce un punct care nu este nici core, nici border este considerat drept zgomot sau outlier.

Figura 5.19 ilustrează principiul de funcționare al algoritmului DBSCAN, evidențiind rolul parametrilor în identificarea punctelor de tip core, border și outlier (anomalie), pe baza densității observațiilor din vecinătatea fiecărui punct.

Procesul începe cu un punct nevizitat. Dacă acesta este core, se creează un cluster și se extinde recursiv cu toate punctele accesibile prin densitate. Dacă este border sau zgomot, algoritmul trece la un alt punct. Procedura se încheie când toate observațiile au fost procesate.

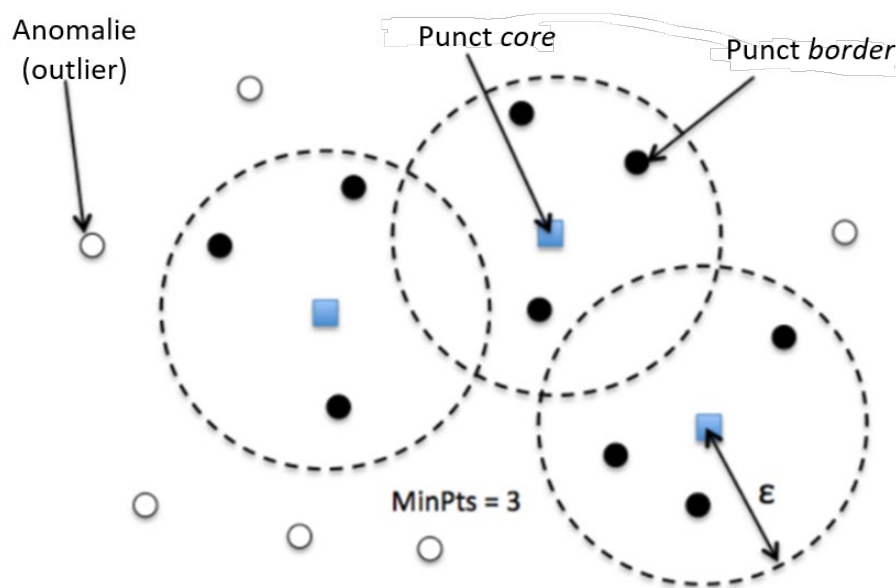


Fig. 5.19. Tipurile de puncte utilizate de DBSCAN

Pași algoritmului DBSCAN sunt următorii:

1. Se selectează arbitrar un punct p
2. Se găsesc toate punctele accesibile din p în raport cu ϵ și MinPts .
3. Dacă p este un punct core, se formează un cluster.
4. Dacă p este un punct de tip border, niciun punct nu este accesibil din punct de vedere al densității din p și DBSCAN vizitează următorul punct.
5. Se continuă procesul până când toate punctele au fost procesate.

5.5.1. Alegerea parametrilor ϵ și MinPts

Alegerea parametrilor influențează decisiv rezultatul. O valoare ϵ prea mică produce multe puncte negrupate și fragmentează clusterelor. O valoare prea mare poate uni regiuni distincte într-un singur cluster. Pentru MinPts se utilizează frecvent valori cel puțin egale cu numărul de dimensiuni plus unu, iar pentru date zgomotoase sunt preferate valori mai mari.

O metodă practică pentru alegerea lui ϵ este graficul k -distance. Pentru fiecare observație se calculează distanța până la al k -lea vecin, unde k este legat de MinPts . Distanțele sunt ordonate, iar o schimbare pronunțată de pantă indică o valoare posibilă

pentru ε . Alegerea măsurii de distanță trebuie realizată înaintea estimării parametrilor și trebuie să fie compatibilă cu tipul datelor.

DBSCAN funcționează bine atunci când clusterelor au densități relativ uniforme. Dacă densitățile diferă mult, un singur ε poate fi prea mic pentru unele grupuri și prea mare pentru altele. În dimensiuni ridicate, noțiunea de vecinătate devine mai puțin relevantă, iar reducerea dimensionalității poate fi necesară.

5.5.2. Exemplu de aplicare DBSCAN

Exemplul pornește de la un set bidimensional pentru care se aleg $\varepsilon = 0,6$ și $\text{MinPts} = 4$. Pentru fiecare punct se determină vecinii aflați la o distanță mai mică sau egală cu ε . Figura 5.20 prezintă punctele de date și afișarea grafică a acestora.

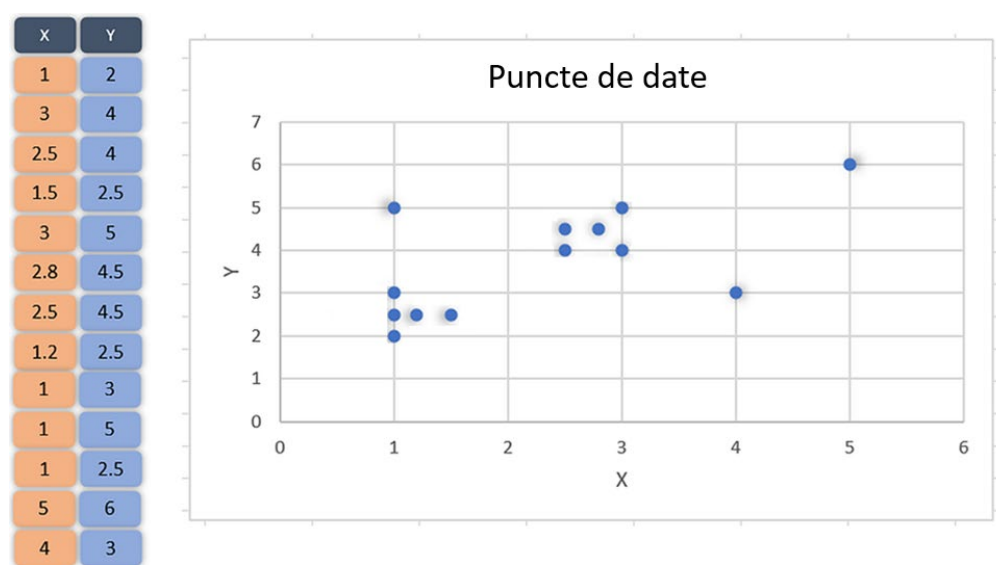


Fig. 5.20. Punctele de date și afișarea grafică

Se consideră apoi primul punct din setul de date, $(1, 2)$, și se calculează distanța acestuia față de toate celelalte observații. Pe baza acestor distanțe se determină vecinătatea punctului pentru valoarea aleasă a parametrului ε , rezultatele fiind prezentate în tabelul următor.

Se observă că punctul $(1, 2)$ are în vecinătatea sa doar două observații, $(1, 2.5)$ și $(1.2, 2.5)$. Deoarece numărul total de puncte din vecinătate este mai mic decât valoarea

$\text{MinPts} = 4$, acesta nu poate fi considerat un punct central (Figura 5.21). Același proces este repetat pentru fiecare observație din setul de date, iar rezultatele obținute sunt sintetizate în tabelul următor, care evidențiază vecinătatea fiecărui punct și permite identificarea punctelor centrale, de frontieră și a anomaliilor.

X	Y	Distanța față de (1,2)
1	2	0
3	4	2.8
2.5	4	2.5
1.5	2.5	0.7
3	5	3.6
2.8	4.5	3.08
2.5	4.5	2.9
1.2	2.5	0.53
1	3	1
1	5	3
1	2.5	0.5
5	6	5.6
4	3	3.1

Punctul (1, 2) are doar alte două puncte în vecinătatea sa (1, 2.5), (1.2, 2.5) - este mai mic decât MinPts

Fig. 5.21. Evaluarea primului punct (1,2)

În tabelul din Figura 5.22 coloana din stânga conține toate punctele din setul de date, iar la dreapta acestora se află punctele de date care se află în vecinătatea lor, adică punctele a căror distanță față de acestea este mai mică sau egală cu valoarea ϵ . Există trei puncte în setul de date, (2.8, 4.5) (1.2, 2.5) (1, 2.5) care au 4 puncte de vecinătate în jurul lor, prin urmare, acestea ar fi numite puncte core și, așa cum am menționat deja, dacă punctul central nu este atribuit niciunui cluster, se formează un nou cluster. Prin urmare, punctul (2.8, 4.5) este atribuit unui nou cluster, Cluster 1, la fel și punctul (1.2, 2.5), Cluster 2. De asemenea, se observă că punctele centrale (1.2, 2.5) și (1, 2.5) au cel puțin un punct de vecinătate comun (1,2), deci sunt atribuite aceluiași cluster. Tabelul arată astfel clasificarea tuturor punctelor de date în puncte centrale (*core*), de frontieră (*border*) și outlier.

Punct	Puncte din vecinătate				
(1,2)	(1.2, 2.5)	(1, 2.5)		Punct <i>border</i>	
(3, 4)	(2.5, 4)	(2.8, 4.5)		Punct <i>border</i>	
(2.5, 4)	(3, 4)	(2.8, 4.5)	(2.5, 4.5)	Punct <i>border</i>	
(1.5, 2.5)	(1.2, 2.5)	(1, 2.5)		Punct <i>border</i>	
(3, 5)	(2.8, 4.5)			Punct <i>border</i>	
(2.8, 4.5)	(3, 4)	(2.5, 4)	(3, 5)	(2.5, 4.5)	Punct <i>core</i> Cluster 1
(2.5, 4.5)	(2.5, 4)	(2.8, 4.5)		Punct <i>border</i>	
(1.2, 2.5)	(1, 2)	(1.5, 2.5)	(1, 3)	(1, 2.5)	Punct <i>core</i> Cluster 2
(1, 3)	(1.2, 2.5)	(1, 2.5)		Punct <i>border</i>	
(1, 5)				Anomalie (<i>Outlier</i>)	
(1, 2.5)	(1, 2)	(1.5, 2.5)	(1.2, 2.5)	(1, 3)	Punct <i>core</i> Cluster 2
(5, 6)				Anomalie (<i>Outlier</i>)	
(4, 3)				Anomalie (<i>Outlier</i>)	

Fig. 5.22. Clasificarea punctelor în core, border și outlier

Rezultatul algoritmului DBSCAN și clusterele rezultate sunt ilustrate în Figura 5.23.

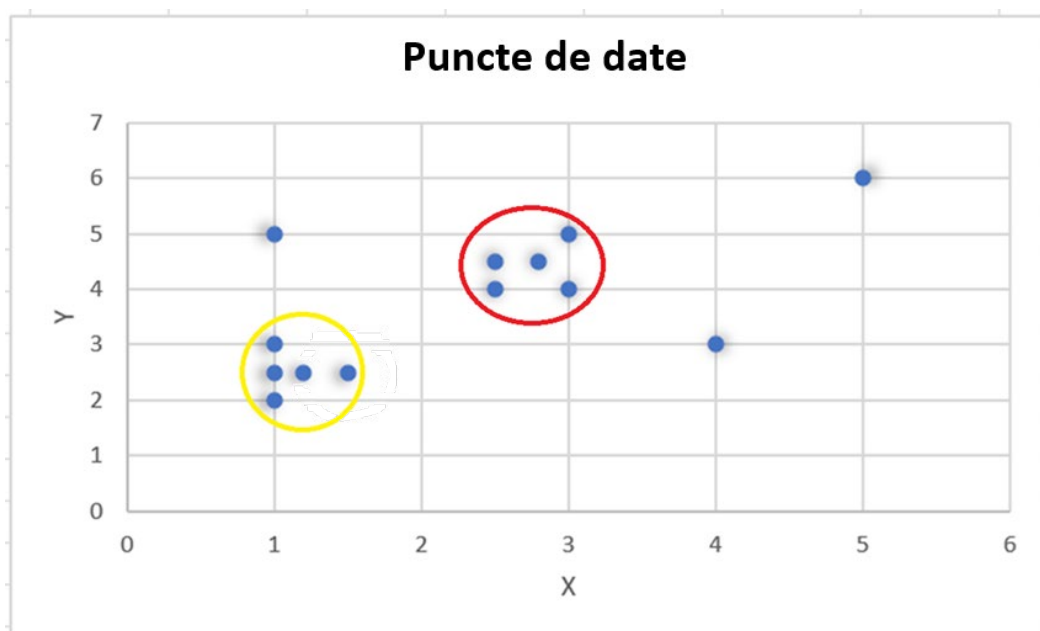


Fig. 5.23. Reprezentarea grafică a clusterelor finale DBSCAN

6. Capitolul 6. Introducere în Rețele Neuronale

Rețelele neuronale artificiale (RNA) constituie o familie de modele de învățare automată inspirate, la nivel de principiu, de modul de organizare și procesare a informației în sistemele neuronale biologice [22]. Într-un model artificial, un număr de unități simple de procesare, denumite neuroni, sunt interconectate prin legături asociate unor ponderi. Fiecare neuron combină informațiile primite, aplică o funcție de activare și transmite rezultatul mai departe. Prin conectarea unui număr mare de astfel de unități pot fi construite modele capabile să reprezinte relații complexe între datele de intrare și ieșirile dorite.

În acest capitol sunt prezentate conceptele fundamentale necesare pentru înțelegerea rețelilor neuronale: inspirația biologică, structura unui neuron artificial, ponderile, bias-ul, funcțiile de activare, rata de învățare, lățimea și adâncimea unei rețele, perceptronul, limitele de decizie, perceptronul multistrat și topologiile feed-forward. Ultima parte este dedicată mecanismului de învățare și algoritmului de backpropagation, inclusiv printr-un exemplu practic urmărit pas cu pas.

6.1. Inspirația biologică a Rețelilor Neuronale Artificiale

Interesul pentru rețele neuronale pornește de la observația că sistemele biologice sunt capabile să realizeze sarcini de o complexitate foarte mare, precum vederea, controlul motor, recunoașterea tiparelor și înțelegerea limbajului natural. Aceste performanțe au sugerat că principiile generale de organizare ale creierului ar putea constitui o sursă de inspirație pentru proiectarea unor sisteme artificiale capabile să învețe din exemple.

Este important însă să facem de la început o distincție între inspirație și reproducere. O rețea neuronală artificială nu este un model fidel al creierului și nici nu urmărește să reproducă toate mecanismele biologice. În schimb, ea păstrează câteva idei generale: existența unor unități de procesare, conectarea acestora prin legături, transmiterea informației și adaptarea conexiunilor în funcție de experiența acumulată.

Ideea centrală nu este copierea creierului, ci preluarea unor principii generale de organizare și procesare a informației. Astfel, neuronul artificial este o abstractizare foarte simplă, bazată în esență pe o sumă ponderată și o funcție de activare.

Din punct de vedere istoric, unul dintre primele repere este anul 1957, când Frank Rosenblatt a introdus perceptronul, un model neuronal concret destinat clasificării. Perceptronul a avut o importanță deosebită deoarece a oferit o formulare matematică simplă a ideii de neuron artificial și a arătat că un model bazat pe ponderi poate învăța o limită de decizie. Ulterior, în 1974, Paul Werbos a introdus ideea utilizării algoritmului de backpropagation pentru rețele multistrat. Această direcție avea să devină fundamentală pentru antrenarea rețelelor neuronale cu straturi ascunse.

Un alt reper este anul 1982, când John Hopfield a prezentat o arhitectură neuronală recurentă, în care neuronii sunt conectați într-o structură ce permite utilizarea rețelei pentru memorare și procesare distribuită. Evoluția domeniului a fost însă întreruptă de perioade de reducere a interesului și finanțării, cunoscute sub denumirea de «iarni ale inteligenței artificiale», precum intervalele 1974–1980 și 1987–2000.

Începând cu anul 2010, rețelele neuronale au cunoscut o revenire majoră, asociată în special cu dezvoltarea învățării profunde (*Deep Learning*). Creșterea puterii de calcul, disponibilitatea unor seturi mari de date și dezvoltarea unor metode mai eficiente de antrenare au permis utilizarea unor rețele cu mai multe straturi și cu un număr mult mai mare de parametri. Astfel, rețelele neuronale au trecut de la modele relativ simple la arhitecturi capabile să învețe reprezentări complexe.

Din punct de vedere tehnic, interesul pentru RNA este justificat de probleme care necesită procesare masiv paralelă și adaptivă. Recunoașterea caracterelor, identificarea tiparelor sau predicția unor stări viitoare sunt exemple de sarcini în care este dificil să fie scrise explicit toate regulile necesare. În locul unei descrieri manuale a tuturor regulilor, rețeaua poate învăța relațiile relevante din exemple.

Există și un al doilea punct de vedere, biologic. Rețelele artificiale pot fi utilizate pentru a reproduce, într-o formă simplificată, anumite principii ale procesării informației biologice și pentru a oferi modele care pot fi studiate în contexte interdisciplinare.

6.2. Structura unei rețele neuronale artificiale

O rețea neuronală artificială este formată dintr-un ansamblu de unități simple de procesare, denumite neuroni, conectate printr-un număr mare de legături ponderate. În funcție de arhitectură, semnalele pot circula numai înainte, de la intrare spre ieșire, sau pot exista conexiuni care permit formarea unor cicluri.

Figura 6.1 prezintă ideea de bază a unei RNA: unități simple de procesare conectate prin legături ponderate. Fiecare conexiune contribuie la calculul ieșirii neuronului către care este orientată.

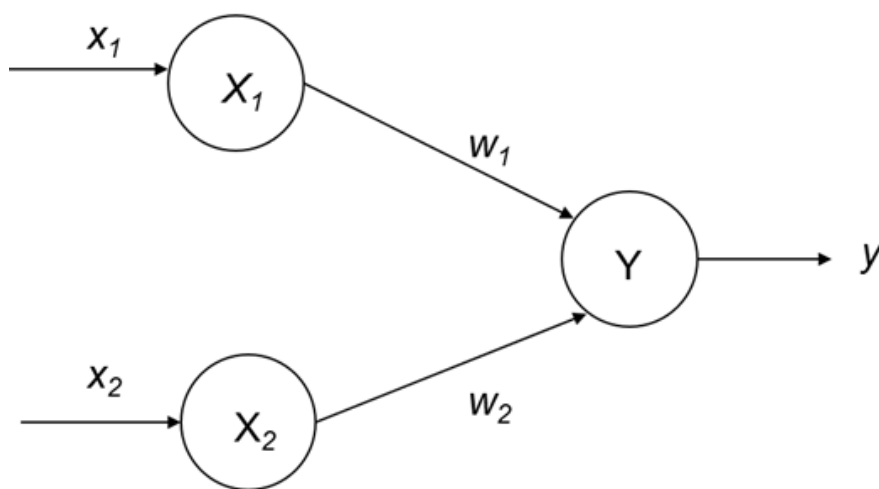


Fig. 6.1. Structura conceptuală a unei rețele neuronale artificiale

O rețea neuronală artificială cuprinde mai multe componente: un set de unități de procesare, o stare de activare pentru fiecare unitate, conexiuni definite prin ponderi, o regulă de propagare, o funcție de activare, intrări externe și o metodă de învățare. Într-un sistem de învățare supervizată se adaugă și informația despre eroare, obținută prin compararea ieșirii modelului cu rezultatul dorit.

O rețea neuronală este alcătuită din neuroni, care reprezintă unitățile elementare de procesare ale rețelei, și din conexiuni între aceștia, fiecărei conexiuni fiindu-i asociată o pondere ce determină contribuția semnalului transmis. Activarea unui neuron reprezintă starea sau valoarea de ieșire a acestuia la un anumit moment al procesului de calcul, în timp ce regula de propagare stabilește modul în care semnalele de intrare sunt

combinate pentru a determina valoarea netă transmisă mai departe. Această valoare este prelucrată prin intermediul unei funcții de activare, care determină ieșirea neuronului, iar regula de învățare stabilește modul în care ponderile și ceilalți parametri ai rețelei sunt modificați pe baza exemplelor de antrenare, astfel încât rețeaua să își îmbunătățească performanța.

Într-o rețea recurentă, semnalul poate ajunge și către neuroni aflați în amonte, ceea ce introduce cicluri. Într-o rețea feed-forward, propagarea are o direcție clară și nu există cicluri.

Figura 6.2 compară conceptual neuronul biologic și modelul artificial. În modelul artificial, complexitatea biologică este redusă la calculul unei sume ponderate, urmat de aplicarea unei funcții de activare.

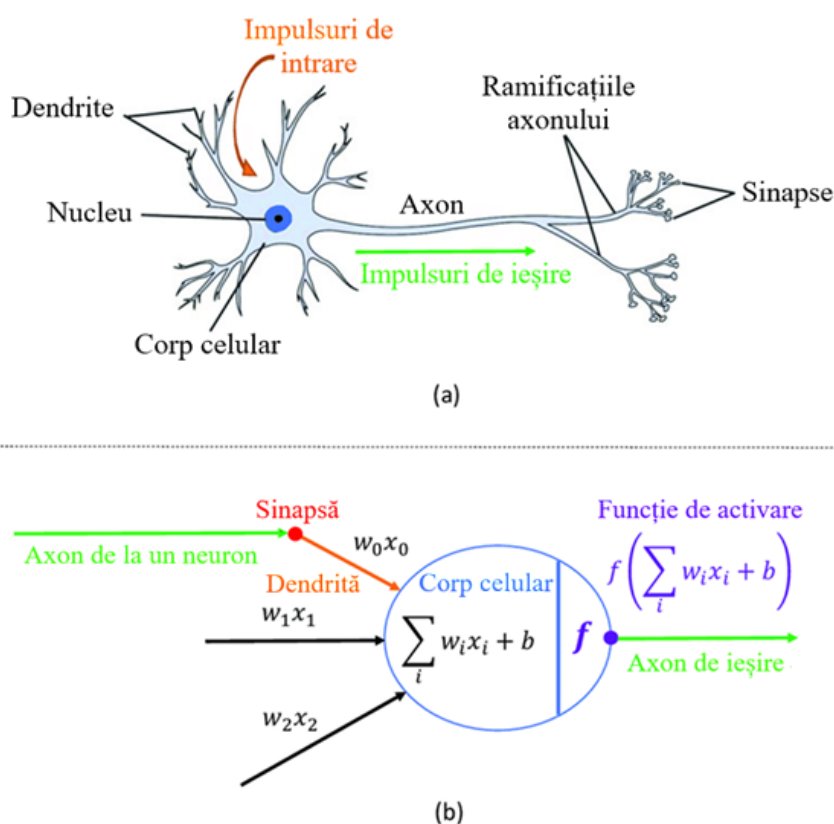


Fig. 6.2. Neuronul biologic și neuronul artificial

Într-o formulare matematică simplificată, pentru intrările $x_1, x_2, \dots, x_n$ și ponderile $w_1, w_2, \dots, w_n$, intrarea netă a neuronului este suma ponderată a intrărilor la care se adaugă bias-ul b :

$$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b \quad (14)$$

Ieșirea neuronului se obține prin aplicarea funcției de activare f :

$$a = f(z) \quad (15)$$

Prin urmare, un neuron nu transmite pur și simplu valoarea unei intrări către următorul neuron. Fiecare semnal este mai întâi ponderat, apoi agregat împreună cu celelalte intrări și, în final, transformat prin funcția de activare. Acest aspect este esențială pentru înțelegerea modului în care o rețea învață.

Ponderile reprezintă parametrii prin care rețeaua exprimă importanța relativă a diferitelor conexiuni. Într-o rețea cu mai mulți neuroni, ponderile dintre două straturi pot fi organizate într-o matrice. Notăția w_{ij} indică, în convenția prezentării, ponderea legăturii de la neuronul i din stratul sursă la neuronul j din stratul destinație. Matricea de ponderi este, astfel, o reprezentare compactă a conexiunilor dintre două straturi.

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ w_3^T \\ \vdots \\ w_n^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1m} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nm} \end{bmatrix}$$

Fig. 6.3. Matricea de ponderi

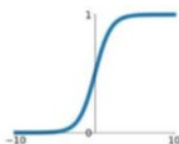
Bias-ul este adăugat la suma ponderată înainte de aplicarea funcției de activare. El permite deplasarea poziției la care neuronul își schimbă răspunsul și crește flexibilitatea modelului [17]. Bias-ul poate fi interpretat ca o variabilă liberă suplimentară. Un bias pozitiv mărește intrarea netă, iar un bias negativ o reduce. O reprezentare utilă este aceea în care bias-ul este tratat ca o pondere asociată unei intrări fixe, egale cu 1. În acest mod, bias-ul poate fi integrat în formalismul general al ponderilor.

Funcția de activare are un rol esențial deoarece introduce, în cazul funcțiilor neliniare, capacitatea rețelei de a modela relații care nu pot fi reprezentate printr-o transformare liniară. Dacă toate straturile unei rețele ar conține numai transformări liniare, compunerea lor ar rămâne o transformare liniară. Prin urmare, adăugarea mai multor straturi nu ar aduce o creștere comparabilă a puterii de reprezentare. Funcțiile neliniare permit fiecărui strat să construiască o reprezentare nouă a datelor.

Figura 6.4 prezintă funcții de activare utilizate pentru a transforma intrarea netă a neuronului. În exemplele din acest capitol, funcția sigmoidă este folosită pentru a urmări explicit calculele de backpropagation.

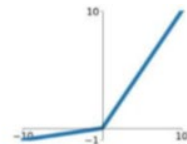
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



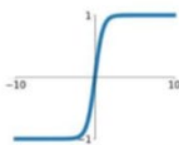
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

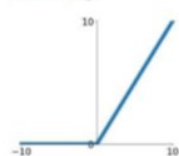


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Fig. 6.4. Funcția de activare non-liniare

Funcția logistică, sau sigmoidă, produce o valoare continuă între 0 și 1 și are forma:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (16)$$

Funcția sigmoidă este utilă pentru exemplele de clasificare binară și pentru demonstrarea calculului gradientului deoarece este derivabilă și are o derivată exprimabilă elegant prin propria valoare:

$$\sigma'(x) = \sigma(x) * [1 - \sigma(x)] \quad (17)$$

În rețelele moderne, alegerea funcției de activare depinde de arhitectură și de problema rezolvată [1]. Pentru straturile ascunse sunt frecvent utilizate funcții precum ReLU și variante ale acesteia, în timp ce funcțiile sigmoidă și softmax apar în special în contexte legate de ieșiri probabilistice și clasificare. Astfel, este important este să fie înțeles rolul funcției de activare și faptul că neliniaritatea este esențială pentru expresivitatea rețelei.

Un alt hiperparametru important este rata de învățare, notată cu α . Aceasta stabilește amplitudinea ajustărilor aplicate ponderilor la fiecare etapă a procesului de antrenare. Dacă rata este prea mare, actualizările pot fi excesive și procesul de optimizare poate deveni instabil; dacă este prea mică, învățarea poate deveni foarte lentă. Rata de învățare este astfel un factor care are influență directă asupra modului în care funcția de eroare converge.

Figura 6.5 ilustrează influența ratei de învățare asupra evoluției pierderii în timpul antrenării rețelei neuronale. O rată foarte mare poate determina creșterea sau oscilația pierderii, una prea mică duce la o convergență lentă, iar o rată de învățare potrivită permite reducerea eficientă a pierderii către o valoare minimă.

Într-o rețea neuronală este util să distingem între lățime și adâncime. Lățimea unui strat este numărul de neuroni din acel strat, iar adâncimea este dată de numărul de straturi. O rețea cu mai multe straturi ascunse este denumită rețea neuronală profundă. Aceste două dimensiuni influențează capacitatea modelului, costul de calcul și modul în care sunt construite reprezentările interne.

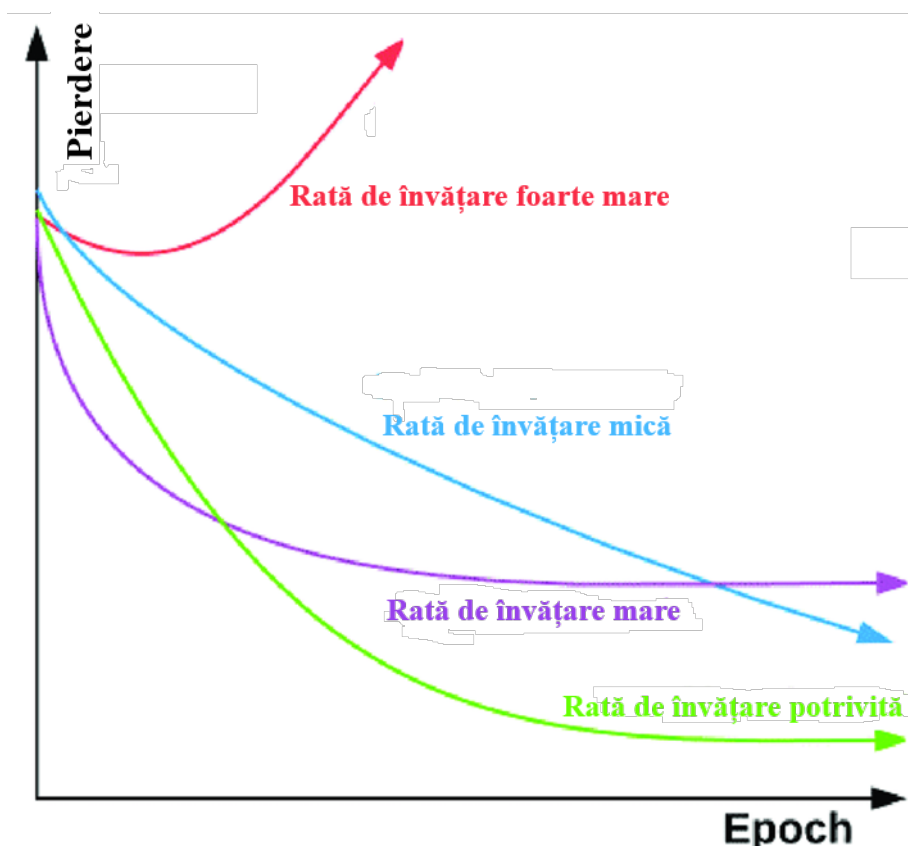


Fig. 6.5. Influența ratei de învățare

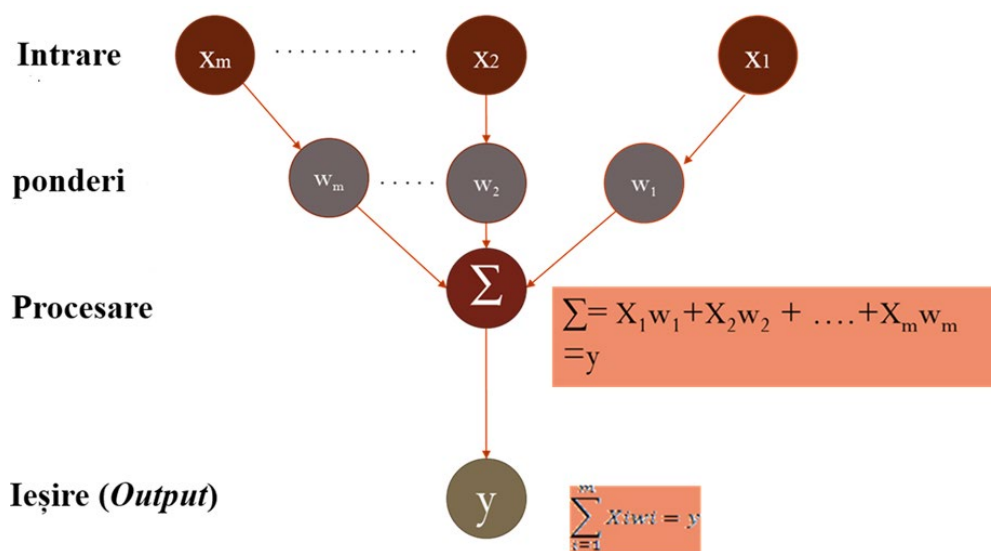


Fig. 6.6. Exemplu simplu de rețea neuronală

Figura 6.6 arată că intrările nu contribuie în mod egal la ieșire. Fiecare intrare este multiplicată cu o pondere, iar rezultatul este apoi combinat și trecut prin funcția de activare.

Un model simplu poate fi interpretat ca o succesiune de operații: preluarea vectorului de intrare, ponderarea componentelor, adăugarea bias-ului, calcularea activării și transmiterea acesteia. Acest mecanism aparent simplu, repetat în multe straturi și pentru foarte mulți neuroni, conduce la o funcție de intrare-ieșire cu un număr mare de parametri.

Numărul de neuroni trebuie ales cu atenție. O rețea cu prea puțini neuroni poate avea o capacitate insuficientă și poate să nu poată surprinde structura datelor. O rețea cu foarte mulți neuroni poate avea o capacitate ridicată, dar poate necesita mai mult timp de antrenare și poate prezenta un risc mai mare de overfitting. În acest caz, modelul poate ajunge să memoreze particularități ale datelor de antrenare și să funcționeze mai slab pe date noi.

6.3. Perceptronul și limitele de decizie

Perceptronul este modelul neuronal clasic asociat cu lucrările lui Rosenblatt și reprezintă una dintre cele mai simple forme de clasificator neuronal. El primește un vector de intrare, calculează suma ponderată a componentelor, adaugă bias-ul și aplică o funcție de activare. Rezultatul poate fi interpretat ca apartenență la una dintre două clase.

Perceptronul este format dintr-un singur neuron artificial și utilizează o funcție de activare cu prag pentru a separa două categorii de observații (Figura 6.7).

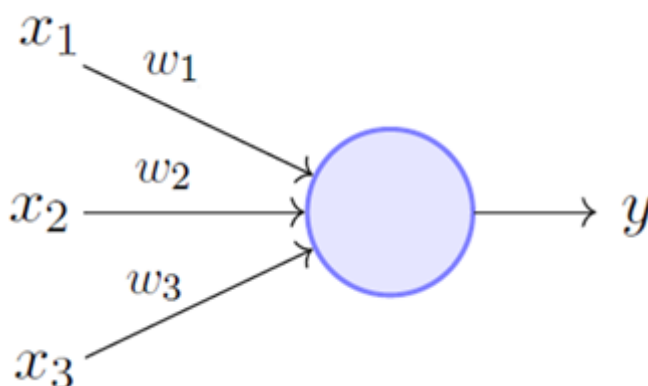


Fig. 6.7. Modelul perceptronului

Pentru un vector de intrare x și un vector de ponderi w , limita de decizie este descrisă de ecuația:

$w^T x + b = 0$	(18)
-----------------	-------------

Dacă valoarea expresiei este pozitivă, neuronul este activat, iar dacă este negativă, neuronul nu este activat. Astfel, modelul împarte spațiul de intrare în două regiuni. În două dimensiuni, limita este o dreaptă; în trei dimensiuni, este un plan; pentru dimensiuni mai mari, este un hiperplan.

Figura 6.8 ilustrează interpretarea geometrică a perceptronului ca clasificator liniar. Vectorul ponderilor este ortogonal la limita de decizie, iar bias-ul contribuie la poziționarea acesteia.

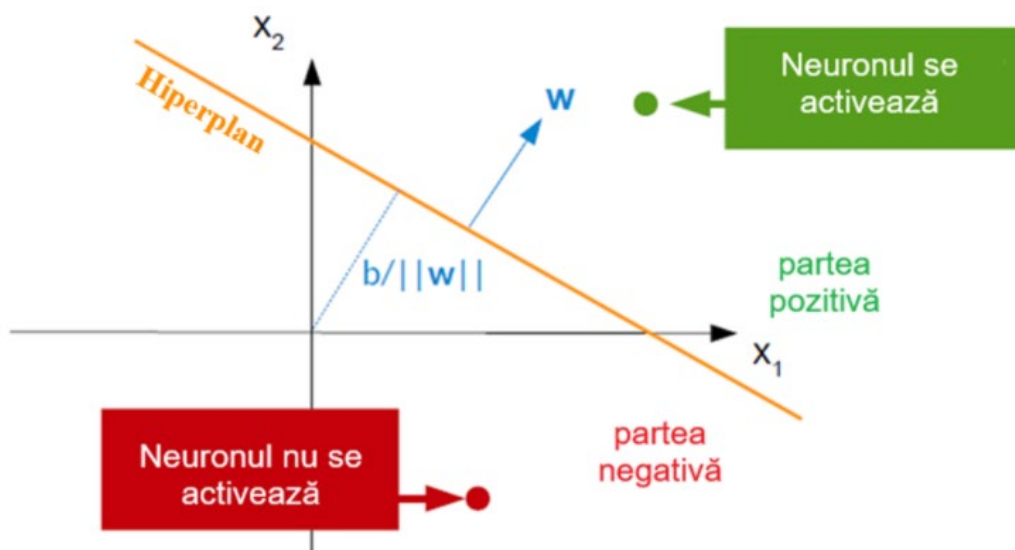


Fig. 6.8. Limita de decizie a unui perceptron

Interpretarea geometrică este utilă deoarece face legătura dintre parametrii matematici și comportamentul modelului. Orientarea limitei de decizie este determinată de vectorul ponderilor, iar poziția acesteia este influențată de bias. Dacă vectorul ponderilor este schimbat, se schimbă orientarea hiperplanului; dacă bias-ul este modificat, hiperplanul se deplasează.

De exemplu, dacă se cunosc ponderile și se dorește ca limita să treacă printr-un anumit punct, bias-ul poate fi determinat prin înlocuirea coordonatelor punctului în ecuația limitei. Această interpretare oferă o metodă intuitivă de a înțelege de ce bias-ul este necesar și de ce acesta nu reprezintă doar un termen auxiliar.

Un perceptron fără straturi ascunde construiește o limită de decizie liniară: dreaptă în două dimensiuni, plan în trei dimensiuni și hiperplan în spații de dimensiune mai mare, așa cum ilustrează Figura 6.9.

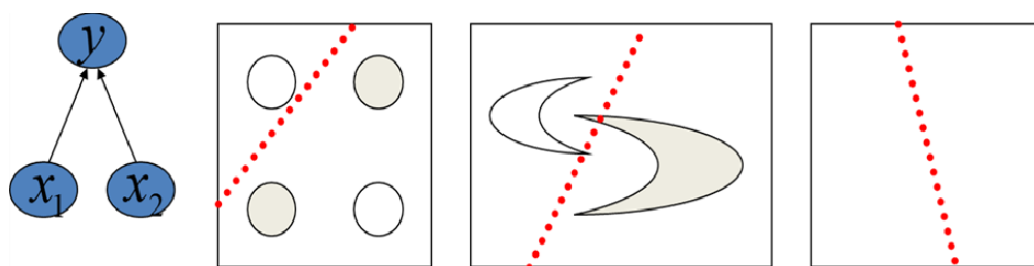


Fig. 6.9. Limita de decizie pentru o rețea fără straturi ascunde

Figura 6.10 ilustrează modul în care introducerea unui strat ascuns permite rețelei neuronale să construiască limite de decizie din ce în ce mai complexe. Spre deosebire de o separare liniară simplă, combinația mai multor neuroni din stratul ascuns poate aproxima limite neliniare, adaptate distribuției punctelor din spațiul de intrare.

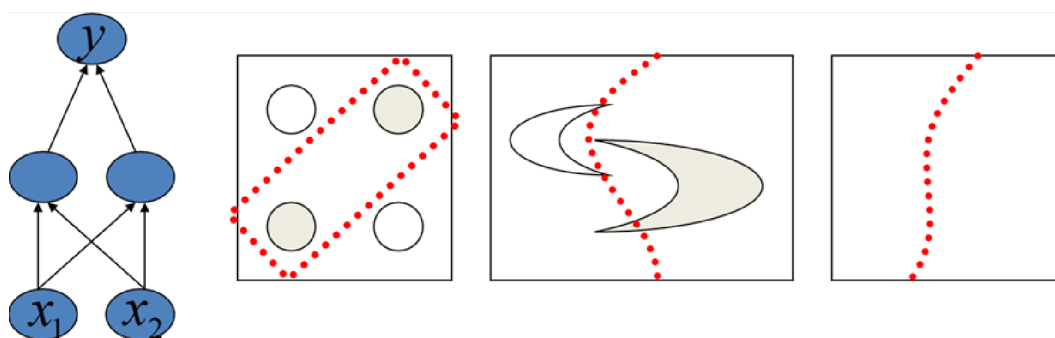


Fig. 6.10. Limita de decizie pentru o rețea cu un strat ascuns

Figura 6.11 ilustrează efectul introducerii unui al doilea strat ascuns în rețeaua neuronală. Prin combinarea succesivă a reprezentărilor învățate în cele două straturi, rețeaua poate construi limite de decizie mult mai complexe și poate separa date care prezintă distribuții neliniare, dificil de modelat cu un singur strat ascuns.

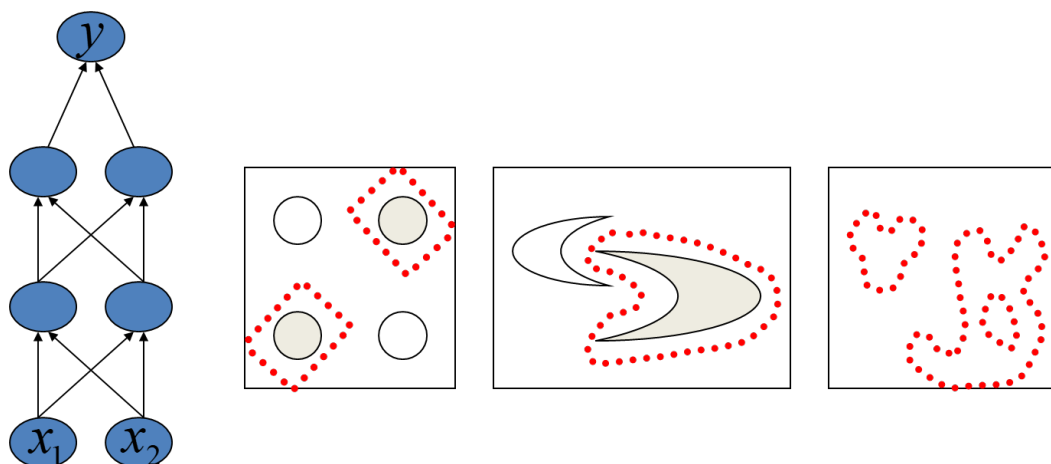


Fig. 6.11. *Limita de decizie pentru o rețea cu două straturi ascunse*

Principalul dezavantaj al perceptronului este limita sa de reprezentare. Un singur neuron poate rezolva probleme liniar separabile, dar nu poate reprezenta o limită de decizie arbitrar de complexă. Funcția booleană AND este un exemplu de problemă liniar separabilă, în timp ce XOR este exemplul clasic de problemă care nu poate fi separată printr-un singur hiperplan.

Figura 6.12 evidențiază limita fundamentală a unui perceptron cu un singur strat: acesta poate învăța numai probleme liniar separabile. Pentru probleme precum XOR este necesară o arhitectură multistrat.

Această limitare nu trebuie interpretată ca un eșec general al ideii de rețea neuronală. Dimpotrivă, ea conduce direct la ideea de a conecta mai mulți neuroni. Un neuron poate construi o limită liniară simplă, iar mai mulți neuroni pot combina aceste limite pentru a forma regiuni mai complexe ale spațiului de intrare.

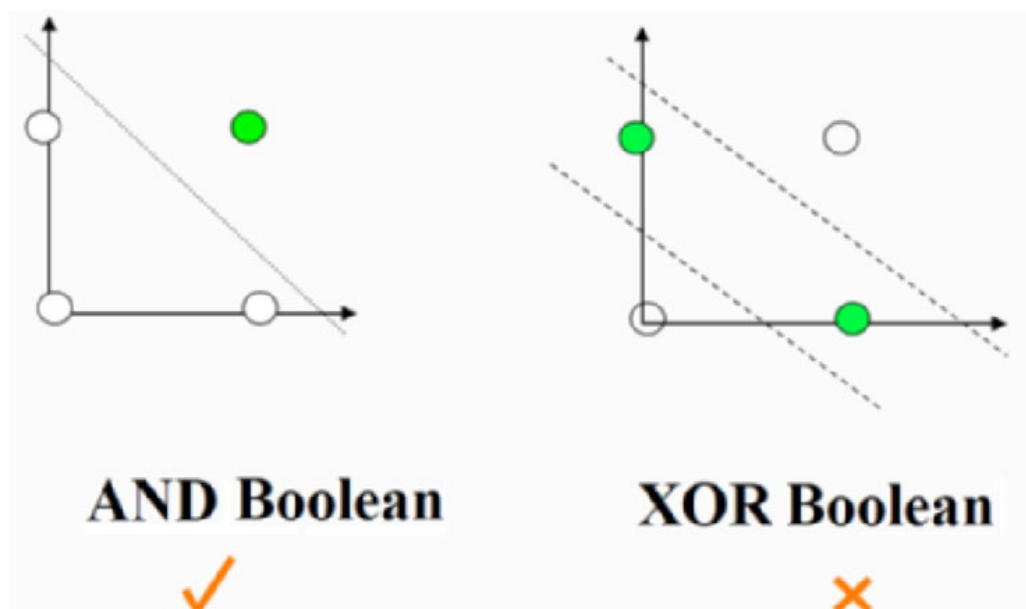


Fig. 6.12. Limitarea perceptronului și problema XOR

Un exemplu geometric este reprezentat de clasificarea unor figuri. Un triunghi poate fi descris prin trei laturi, iar fiecare latură poate fi aproximată printr-o limită liniară. Un dreptunghi necesită patru laturi și poate fi construit prin combinarea a patru limite. Un cerc nu are laturi drepte, astfel încât limita sa poate fi aproximată prin mai multe segmente liniare. Cu cât se utilizează mai multe segmente, cu atât aproximația poate deveni mai precisă.

Figura 6.13 ilustrează ideea că mai mulți neuroni pot aproxima forme geometrice prin combinarea unor limite de decizie liniare. Formele cu frontiere mai complexe necesită, în general, mai multe unități pentru o aproximație mai precisă.

Această interpretare oferă o intuiție foarte bună asupra rolului stratului ascuns. Neuronii dintr-un strat ascuns nu trebuie priviți numai ca unități care produc valori intermediare, ci ca elemente care construiesc reprezentări și regiuni intermediare ale spațiului de intrare. Stratul următor poate combina aceste reprezentări pentru a obține o decizie finală.

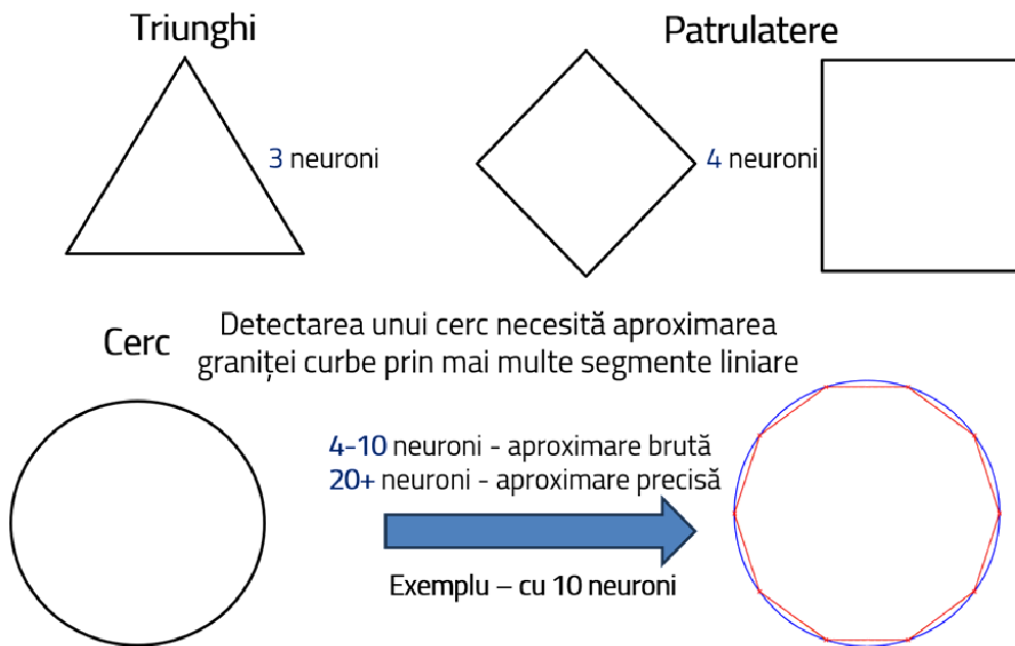


Fig. 6.13. Aproximarea geometrică folosind mai mulți neuroni

6.4. Topologiile Rețelelor Neuronale

Pentru depășirea limitărilor perceptronului simplu se utilizează rețele cu unul sau mai multe straturi ascunse. Un perceptron multistrat (*Multilayer Perceptron* – MLP) este alcătuit, în forma sa de bază, dintr-un strat de intrare, unul sau mai multe straturi ascunse și un strat de ieșire. Straturile ascunse sunt numite astfel deoarece rezultatele lor intermediare nu sunt observate direct din exteriorul rețelei.

Figura 6.14 prezintă structura generală a unui MLP, cu strat de intrare, straturi ascunse și strat de ieșire. Straturile ascunse permit construirea unor reprezentări intermediare.

Fiecare strat poate avea un număr diferit de neuroni, iar în principiu pot fi utilizate funcții de activare diferite.

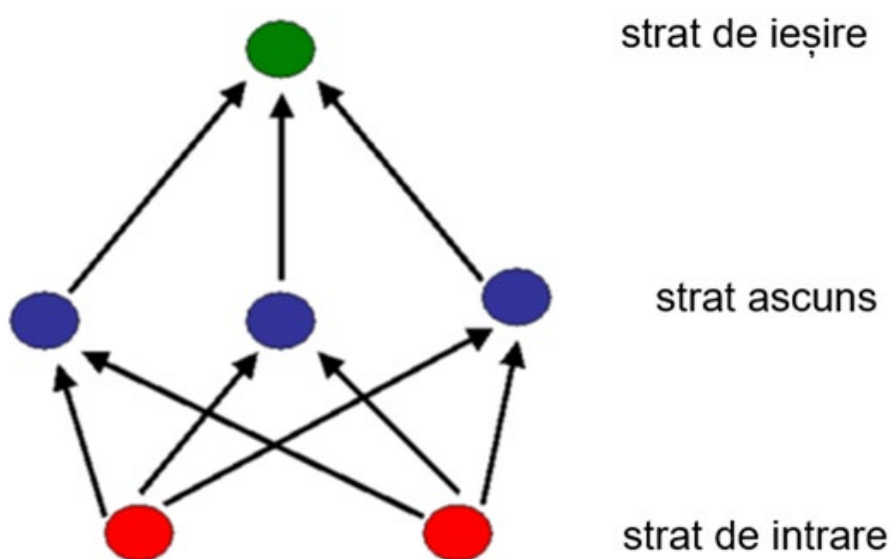


Fig. 6.14. *Perceptronul multistrat*

Figura 6.15 prezintă trei tipuri de arhitecturi ale rețelelor neuronale: rețele complet conectate, în care neuronii sunt conectați dens între ei, rețele feed-forward, în care informația se propagă într-o singură direcție, de la intrare spre ieșire, și rețele recurente, care includ conexiuni de feedback și permit utilizarea informației din stările anterioare.

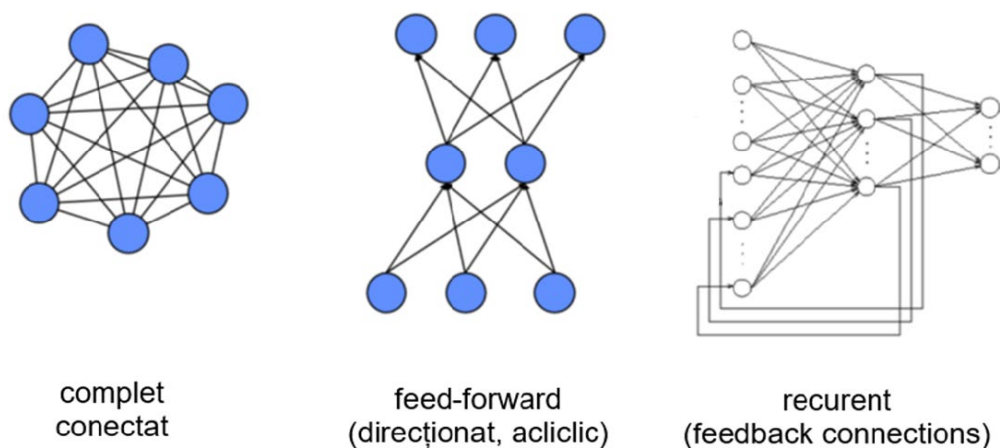


Fig. 6.15. *Structura și activările unui MLP*

Într-o rețea feed-forward cu un singur strat de calcul, stratul de intrare este proiectat direct în stratul de ieșire. Într-o rețea feed-forward multistrat există unul sau mai multe straturi ascunse, iar intrarea este transformată succesiv. Această structură permite modelului să construiască reprezentări din ce în ce mai complexe.

Așa cum arată și Figura 6.16, într-o rețea feed-forward nu există cicluri. Informația este transmisă într-o singură direcție, iar neuronii sunt organizați în straturi.

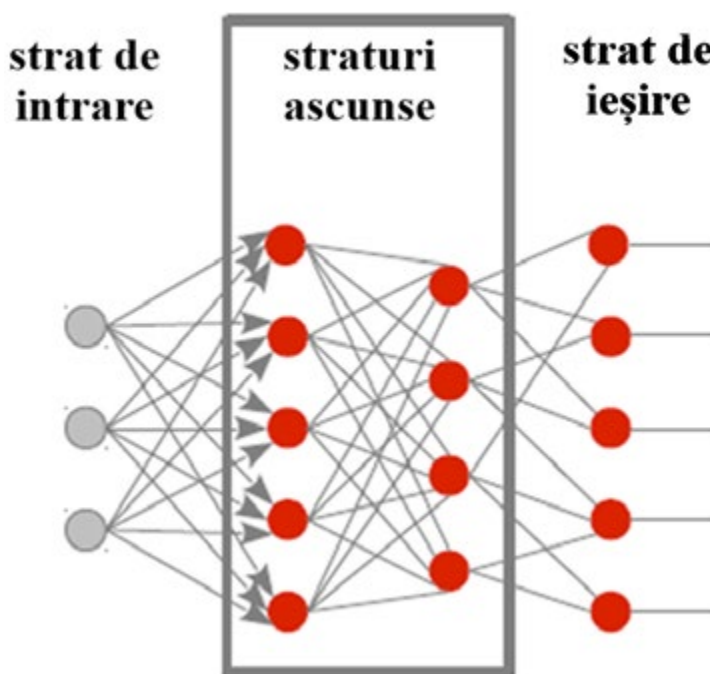


Fig. 6.16. *Rețea neuronală feed-forward*

Din punct de vedere conceptual, o rețea poate fi caracterizată prin lățime și adâncime. Lățimea este legată de numărul de neuroni dintr-un strat, în timp ce adâncimea reflectă numărul de straturi. O creștere a lățimii poate permite reprezentarea mai multor componente ale unei probleme la același nivel, în timp ce creșterea adâncimii permite compunerea succesivă a unor transformări.

În condiții matematice adecvate și cu suficienți neuroni, o rețea cu un strat ascuns poate aproxima o clasă foarte largă de funcții continue. Această proprietate este importantă pentru a înțelege de ce rețelele neuronale au o putere de reprezentare

ridicată. Totuși, această teoremă nu spune câți neuroni sunt necesari pentru o problemă concretă și nici că antrenarea va găsi automat cea mai bună soluție.

Pe măsură ce sunt adăugate straturi, rețeaua poate construi reprezentări ierarhice. Straturile apropiate de intrare pot învăța structuri simple, iar cele mai profunde pot combina aceste structuri în reprezentări mai complexe. În arhitecturile specializate, această idee este exploatată în mod diferit, în funcție de tipul datelor și de conexiunile utilizate.

O consecință importantă este aceea că alegerea arhitecturii devine o problemă de proiectare. Creșterea numărului de neuroni sau a numărului de straturi ale rețelei nu duce în mod implicit la îmbunătățirea performanței modelului. O rețea trebuie să aibă suficientă capacitate pentru problema dată, dar și să poată fi antrenată și să generalizeze la date noi.

Pentru un model prea simplu apare riscul de underfitting: eroarea rămâne mare deoarece modelul nu are capacitatea necesară. Pentru un model prea complex apare riscul de overfitting: eroarea pe datele de antrenare poate deveni foarte mică, în timp ce performanța pe date nevăzute se poate deteriora.

6.5. Mecanismul de învățare și algoritmul de backpropagation

După stabilirea arhitecturii, rămâne problema esențială a învățării ponderilor și bias-urilor. În cazul antrenării supervizate, setul de date este alcătuit din perechi formate din vectori de intrare și ieșiri așteptate. Rețeaua primește un vector x , produce o predicție $\hat{y}$, iar această predicție este comparată cu ținta y . Parametrii sunt apoi modificați astfel încât predicția viitoare să fie mai apropiată de țintă [6].

Figura 6.17 introduce cadrul de antrenare utilizat în acest capitol: exemple de intrare, ținte și o rețea MLP cu funcție de activare sigmoidă.

Pentru clasificarea cu mai multe clase, prezentarea utilizează ideea de One-Hot Encoding. Se consideră câte un neuron de ieșire pentru fiecare clasă, iar vectorul-țintă are valoarea 1 pe poziția clasei corecte și 0 pe celelalte poziții. De exemplu, pentru recunoașterea cifrei 4, vectorul-țintă poate fi reprezentat printr-un vector cu zece componente, în care a patra componentă este 1, iar celelalte sunt 0.

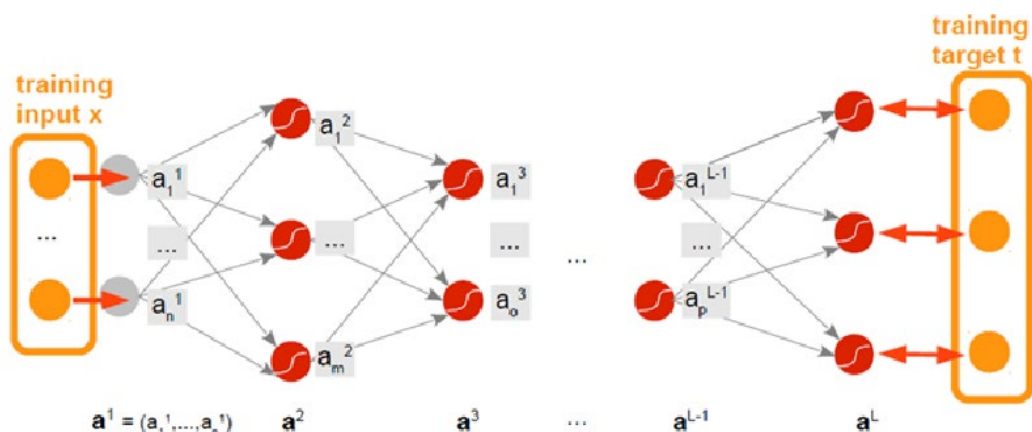


Fig. 6.17. Mecanismul de învățare supervizată

Antrenarea alternează două faze principale. În faza de propagare înainte, un eșantion este introdus în rețea, iar activările sunt calculate succesiv pentru toate straturile. În faza backward, rezultatul final este comparat cu ținta, iar informația despre eroare este utilizată pentru actualizarea ponderilor și bias-urilor din toate straturile.

Figura 6.18 sintetizează cele două faze ale antrenării: propagarea datelor către ieșire și propagarea informației despre eroare în sens invers.

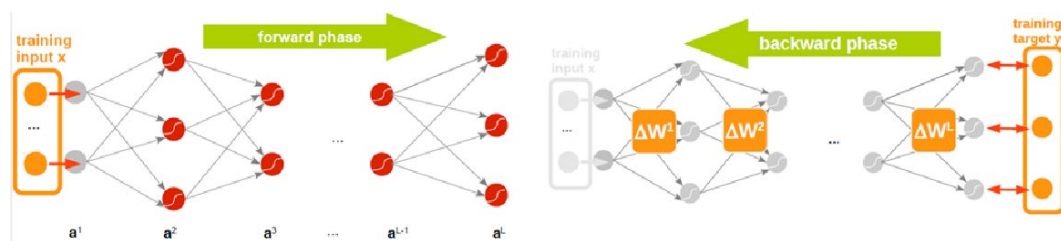


Fig. 6.18. Forward pass și backward pass

6.5.1. Notății și modelul computațional

Pentru o rețea cu L straturi, notăm cu a^l vectorul activărilor stratului l , cu b^l vectorul bias-urilor, cu z^l vectorul intrărilor nete și cu W^l matricea ponderilor dintre stratul $l-1$ și stratul l . În această notație, operația fundamentală a unui strat poate fi scrisă:

$$z^l = W^l * a^{l-1} + b^l \quad (19)$$

După calcularea lui z^l , funcția de activare f este aplicată componentă cu componentă:

$$a^l = f(z^l) \quad (20)$$

În acest fel, propagarea printr-o rețea multistrat poate fi privită ca o succesiune de transformări liniare și neliniare. Ultimul vector de activări constituie ieșirea rețelei:

$$\hat{y} = a^L \quad (21)$$

Avantajul notației matriciale este că permite exprimarea compactă a operațiilor și, mai important, implementarea lor eficientă. Într-o rețea cu sute sau mii de neuroni, calcularea fiecărei conexiuni separat ar fi nepractică. Operațiile matriciale permit utilizarea procesoarelor și acceleratoarelor moderne pentru a calcula simultan numeroase produse și sume.

6.5.2. Funcția de pierdere și gradient descent

Pentru a putea modifica parametrii, este necesară o măsură numerică a erorii. Această măsură este furnizată de funcția de pierdere L . În funcție de problemă, pot fi utilizate diferite funcții de pierdere. În acest capitol se utilizează eroarea pătratică:

$$L = \frac{1}{2} * (y - \hat{y})^2 \quad (22)$$

Interpretarea este simplă: cu cât predicția $\hat{y}$ este mai apropiată de ținta y , cu atât valoarea pierderii este mai mică. Factorul $1/2$ este convenabil pentru derivare și nu schimbă poziția minimului.

Scopul antrenării este minimizarea pierderii în raport cu parametrii rețelei. Dacă w este un parametru, gradientul $\partial L / \partial w$ indică direcția în care pierderea crește cel mai repede. Pentru a reduce pierderea, actualizarea se face în direcția opusă gradientului:

$$w = w - \alpha * \frac{\partial L}{\partial w} \quad (23)$$

Această regulă stă la baza metodei *gradient descent*. În practică, gradientul este adesea calculat pe mini-loturi de exemple, iar actualizarea este repetată de multe ori

6.5.3. Backpropagation și regula lanțului

Dificultatea apare atunci când rețeaua are straturi ascunse. O modificare a unei ponderi din primul strat poate influența activarea unui neuron ascuns, care influențează un alt neuron, iar efectul se propagă până la ieșirea finală. Pentru a determina cât de mult contribuie acea pondere la pierdere, trebuie urmărite aceste dependențe.

Backpropagation rezolvă problema prin aplicarea repetată a regulii lanțului. Dacă o variabilă depinde de o altă variabilă, iar aceasta depinde de o a treia variabilă, derivata funcției compuse poate fi calculată prin produsul derivatelor intermediare. În formă simplă, dacă $z=f(y)$ și $y=g(x)$, atunci:

$$\frac{\delta z}{\delta x} = \frac{\delta z}{\delta y} * \frac{\delta y}{\delta x} \quad (24)$$

Într-o rețea neuronală, exact această idee este aplicată asupra structurii de calcul. Se pornește de la funcția de pierdere, se calculează gradientul pentru stratul de ieșire, apoi se continuă către straturile ascunse. Astfel, aceeași structură de calcul care produce ieșirea este parcursă în sens invers pentru a determina contribuția parametrilor la eroare.

Procesul de antrenare presupune următoarele etape principale:

1. Forward Pass: determinarea ieșirilor – se calculează activările pentru toate straturile și ieșirea finală.
2. Calcularea pierderii: evaluarea erorii – se determină pierderea L utilizând funcția de pierdere.
3. Backward Pass: determinarea gradientilor – se utilizează regula lanțului (chain rule) pentru a calcula gradientii pornind de la stratul de ieșire și propagându-se înapoi prin fiecare strat până la primul strat.
4. Actualizarea parametrilor – se ajustează ponderile și bias-urile utilizând gradientii calculați.

Pentru stratul de ieșire, gradientul este calculat pornind de la funcția de pierdere și derivata funcției de activare. Dacă δ_j^L reprezintă termenul de eroare al neuronului j din stratul final, pentru funcția pătratică se poate scrie, în forma utilizată în exemplu:

$$\delta_j^L = (y_j - a_j^L) * f'(z_j^L) \quad (25)$$

Semnul acestei expresii depinde de convenția utilizată pentru definirea termenului de eroare și a regulii de actualizare. Termenul este folosit împreună cu o regulă de actualizare aditivă, $\Delta w = \alpha \delta a$. Este important ca cele două convenții să fie utilizate consecvent.

Pentru un neuron dintr-un strat ascuns, influența sa asupra pierderii este indirectă: neuronul contribuie la activările stratului următor, iar acestea contribuie la ieșirea finală. De aceea, termenul de eroare al unui strat ascuns este calculat folosind termenii de eroare ai stratului următor și ponderile conexiunilor dintre straturi.

$$\delta^k = (W^{l+1})^T * \delta^{l+1} \odot f'(z^l) \quad (26)$$

Această relație poate fi interpretată intuitiv astfel: gradientul primit din amonte este combinat cu ponderile prin care neuronul influențează stratul următor, iar rezultatul este multiplicat cu derivata funcției de activare a neuronului curent.

Astfel, eroarea unui neuron ascuns este determinată de influența sa asupra stratului următor și de derivata funcției sale de activare.

6.5.4. Actualizarea ponderilor și bias-urilor

După calcularea termenilor de eroare, se determină gradientii parametrilor. Pentru o conexiune care pleacă de la neuronul i și ajunge la neuronul j , formula de calcul este:

$$\Delta w_{ij} = \alpha * \delta_j * a_i \quad (27)$$

Noua pondere se obține apoi prin:

$$x_{ij}^{nou} = w_{ij} + \Delta w_{ij} \quad (28)$$

În mod similar, bias-ul este actualizat pe baza termenului de eroare al neuronului și a ratei de învățare. Esența algoritmului este că fiecare parametru este ajustat proporțional cu contribuția sa la eroarea observată. Astfel, conexiunile care au o influență mai mare asupra erorii primesc, în general, actualizări mai mari.

6.6. Exemplu practic

Pentru a înțelege mecanismul de backpropagation, prezentarea utilizează o rețea foarte mică, cu două intrări, un strat ascuns format din doi neuroni și un neuron de ieșire. Exemplul este suficient de simplu pentru a putea urmări pas cu pas toate operațiile, dar păstrează structura fundamentală a unei rețele multistrat.

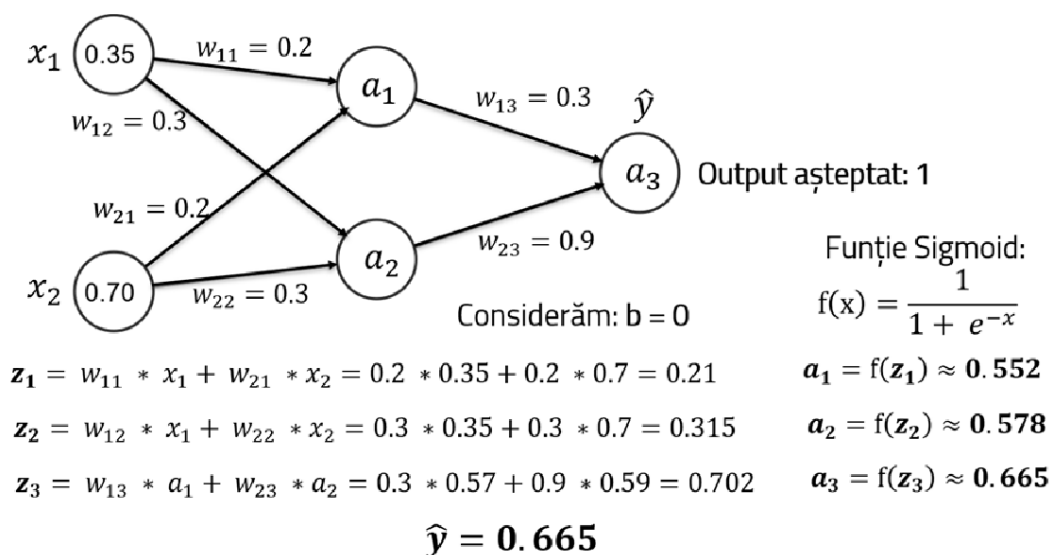


Fig. 6.19. Rețeaua utilizată în exemplul din această secțiune

Figura 6.19 prezintă rețeaua cu două intrări $x_1=0,35$ și $x_2=0,70$, doi neuroni ascunși și un neuron de ieșire. Bias-ul este considerat $b=0$, iar funcția de activare este sigmoidă. Ieșirea așteptată este 1.

Valorile inițiale ale ponderilor sunt $w_{11}=0,2$, $w_{12}=0,3$, $w_{21}=0,2$, $w_{22}=0,3$, $w_{13}=0,3$ și $w_{23}=0,9$. Pentru claritate, indicii pot fi interpretați ca referindu-se la conexiunea dintre un neuron sursă și unul destinație, conform convenției din acest capitol.

6.6.1. Propagare înainte (*Forward pass*)

Primul pas constă în calcularea intrării nete a primului neuron ascuns. Deoarece bias-ul este zero, formula devine:

$$z_1 = w_{11}x_1 + w_{21}x_2 = 0.2 * 0.35 + 0.2 * 0.7 = 0.21$$

Aplicând funcția sigmoidă:

$$a_1 = \sigma(0.21) \approx 0.552$$

Pentru al doilea neuron ascuns:

$$z_2 = w_{12}x_1 + w_{22}x_2 = 0.3 * 0.35 + 0.3 * 0.7 = 0.315$$

$$a_2 = \sigma(0.315) \approx 0.578$$

Aceste două activări constituie intrările neuronului de ieșire. Se calculează mai întâi intrarea netă:

$$z_3 = w_{13} * a_1 + w_{23} * a_2 \tag{29}$$

$$z_3 = 0.3 * 0.552 + 0.9 * 0.578 = 0.702$$

Aplicând funcția sigmoidă se obține ieșirea rețelei:

$$\hat{y} = a_3 = \sigma(z_3) \approx 0.665$$

Prin urmare, pentru intrarea (0,35; 0,70), rețeaua produce o predicție de aproximativ 0,665, în timp ce valoarea-țintă este 1. Exemplul arată clar că rețeaua nu produce direct eticheta finală, ci o valoare rezultată din compunerea succesivă a transformărilor efectuate de neuroni.

6.6.2. Calcularea pierderii

Pentru a cuantifica diferența dintre predicție și țintă, se utilizează funcția de pierdere pătratică:

$$L = \frac{1}{2} * (y - \hat{y})^2$$

$$L = \frac{1}{2} * (1 - 0.665)^2 \approx 0.0561$$

Valoarea pierderii nu este încă suficient de mică pentru ca modelul să producă exact rezultatul dorit. Important este însă că această valoare poate fi folosită pentru a determina în ce direcție trebuie modificate ponderile.

Figura prezintă calculul pierderii pornind de la valoarea-țintă 1 și ieșirea obținută de rețea, aproximativ 0,665.

6.6.3. Backpropagation pentru stratul de ieșire

Pentru funcția sigmoidă se utilizează proprietatea:

$$f'(x) = f(x)[1 - f(x)] \quad (30)$$

Pentru neuronul de ieșire, cu $a_3 \approx 0,665$:

$$f'(a_3) = 0.665 * (1 - 0.665) \approx 0.2228$$

Termenul de eroare este:

$$\delta_3 = (y - a_3) * f'(a_3) \quad (31)$$

$$\delta_3 = (1 - 0.665) * 0.2228 \approx 0.0746$$

Acest termen exprimă, în convenția utilizată în exemplu, direcția și amplitudinea ajustării necesare la ieșire. Pentru rata de învățare $\alpha=0,5$, modificarea primei ponderi de ieșire este:

$$\Delta w_{13} = \alpha * \delta_3 * a_1 = 0.5 * 0.0746 * 0.552 \approx 0.0206$$

$$w_{13}^{nou} = 0.5 + 0.0206 \approx 0.3206$$

Pentru a doua conexiune:

$$\Delta w_{23} = 0.5 * 0.0746 * 0.578 \approx 0.0216$$

$$w_{23}^{nou} = 0.9 + 0.0216 \approx 0.9216$$

Figura 6.20 prezintă calculul termenului de eroare pentru neuronul de ieșire și actualizarea celor două ponderi care leagă stratul ascuns de ieșire.

Pentru funcția Sigmoid: $f'(x) = f(x) * (1 - f(x))$

$$f'(a_3) = a_3(1 - a_3) = 0.665 * (1 - 0.665) \approx 0.2228$$

$$\delta_5 = (y - a_3) * f'(a_3) = (1 - 0.665) * 0.2228 \approx 0.0746$$

$$\Delta w_{i3} = \alpha * \delta_5 * a_i$$

Considerăm: $\alpha = 0.5$

$$\Delta w_{13} = 0.5 * 0.0746 * 0.552 \approx 0.0206$$

$$w_{13}^{nou} = w_{13} + \Delta w_{13} = 0.3 + 0.0206 \approx 0.3206$$

$$\Delta w_{23} = 0.5 * 0.0746 * 0.578 \approx 0.0216$$

Fig. 6.20. Actualizarea ponderilor stratului de ieșire

6.6.4. Backpropagation pentru stratul ascuns

După actualizarea stratului de ieșire, eroarea trebuie propagată către neuronii ascunși. Fiecare neuron ascuns influențează ieșirea prin propria activare și prin ponderea conexiunii către neuronul final. Prin urmare, termenul de eroare al fiecărui neuron ascuns depinde de termenul de eroare al neuronului de ieșire, de ponderea conexiunii și de derivata funcției de activare.

Pentru primul neuron ascuns, termenul poate fi exprimat conceptual prin:

$$\delta_1 = \delta_3 * w_{13} * f'(a_1) \quad (32)$$

Pentru al doilea neuron:

$$\delta_2 = \delta_3 * w_{23} * f'(a_2) \quad (33)$$

Calculul exact urmează aceleași etape ca în stratul de ieșire: se determină derivata locală a funcției de activare și se combină cu semnalul de eroare propagat din amonte. Astfel, backpropagation nu presupune recalcularea de la zero a întregii rețele, ci reutilizează informația calculată deja la stratul următor.

Figura 6.21 ilustrează propagarea erorii de la neuronul de ieșire către neuronii din stratul ascuns, folosind ponderile conexiunilor și derivatele funcțiilor de activare.

$$f'(a_1) = a_1(1 - a_1) = 0.552 * (1 - 0.552) \approx 0.247$$

$$f'(a_2) = a_2(1 - a_2) = 0.578 * (1 - 0.578) \approx 0.244$$

$$\delta_3 = f'(a_1) * (\delta_5 * w_{13}) = 0.247 * (0.0746 * 0.3) \approx 0.0055$$

$$\delta_4 = f'(a_2) * (\delta_5 * w_{23}) = 0.244 * (0.0746 * 0.9) \approx 0.0164$$

$$\Delta w_{ij} = \alpha * \delta_{j+2} * x_i$$

$$\Delta w_{11} = \alpha * \delta_3 * x_1 = 0.5 * 0.0055 * 0.35 \approx 0.00096$$

$$w_{11}^{nou} = w_{11} + \Delta w_{11} = 0.2 + 0.00096 \approx 0.20096$$

Fig. 6.21. *Backpropagation pentru stratul ascuns*

Odată determinați termenii δ_i și δ_j , se pot actualiza ponderile care leagă intrările de stratul ascuns. De exemplu, pentru o conexiune care intră în primul neuron ascuns, modificarea este calculată cu aceeași regulă:

$$\Delta w = \alpha * \delta * x \quad (34)$$

În acest fel sunt ajustate succesiv toate ponderile și bias-urile rețelei. După finalizarea acestei etape, rețeaua este pregătită pentru un nou forward pass.

Figura 6.22 prezintă continuarea procesului de învățare după calcularea termenilor de eroare pentru neuronii ascunși. Ponderile sunt modificate pentru ca următoarea predicție să fie mai apropiată de țintă.

$$\Delta w_{11} = \alpha * \delta_3 * x_1 = 0.5 * 0.0055 * 0.35 \approx 0.00096$$

$$w_{11}^{nou} = w_{11} + \Delta w_{11} = 0.2 + 0.00096 \approx 0.20096$$

$$\Delta w_{12} = \alpha * \delta_4 * x_1 = 0.5 * 0.0164 * 0.35 \approx 0.00287$$

$$w_{12}^{nou} = w_{12} + \Delta w_{12} = 0.3 + 0.00287 \approx 0.30287$$

$$\Delta w_{22} = \alpha * \delta_4 * x_1 = 0.5 * 0.0164 * 0.7 \approx 0.00574$$

$$w_{22}^{nou} = w_{22} + \Delta w_{22} = 0.3 + 0.00574 \approx 0.30574$$

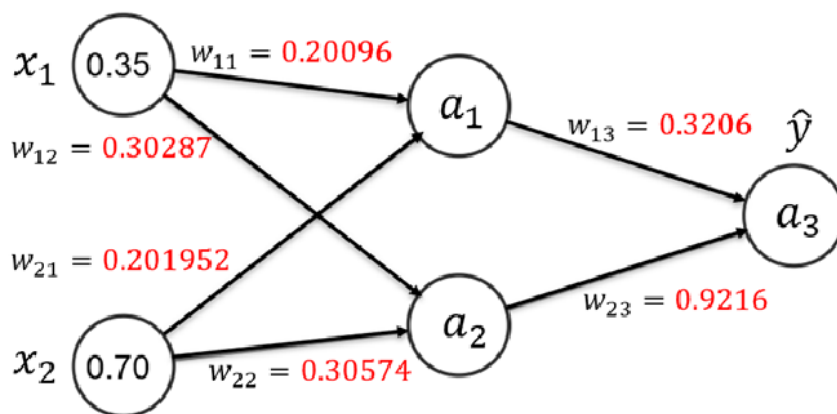


Fig. 6.22. Actualizarea parametrilor din stratul ascuns.

Un singur pas de backpropagation modifică parametrii doar puțin. Pentru ca rețeaua să învețe o relație generală, procesul este repetat pentru numeroase eșantioane și, de regulă, pentru mai multe parcurgeri ale setului de antrenare. Fiecare trecere completă prin setul de antrenare este denumită epocă (*epoch*).

Pe parcursul antrenării se urmărește, de regulă, evoluția funcției de pierdere. Dacă procesul de optimizare este bine configurat, pierderea pe setul de antrenare tinde să scadă. Totuși, scăderea continuă a pierderii pe datele de antrenare nu garantează îmbunătățirea performanței pe date noi. Din acest motiv, seturile de antrenare, validare și testare trebuie tratate distinct.

În practică, rata de învățare, inițializarea parametrilor, dimensiunea rețelei și alegerea funcțiilor de activare pot influența puternic procesul de antrenare. În rețelele profunde pot apărea probleme precum *vanishing gradients* și *exploding gradients*, iar metode moderne de optimizare și regularizare sunt folosite pentru a îmbunătăți stabilitatea antrenării. Backpropagation rămâne însă mecanismul fundamental prin care gradientul este calculat și pus la dispoziția algoritmului de optimizare.

7. Capitolul 7. Rețele Neuronale Convoluționale

Rețelele neuronale convoluționale (Convolutional Neural Networks – CNN sau RNC) reprezintă una dintre cele mai importante arhitecturi de învățare profundă pentru prelucrarea datelor cu structură spațială. Ele au fost dezvoltate pornind de la observația că imaginile nu sunt simple liste de valori numerice, ci structuri bidimensionale în care vecinătatea și poziția elementelor au o importanță esențială.

În acest capitol sunt prezentate principiile care stau la baza rețelelor convoluționale, pornind de la reprezentarea imaginilor ca date numerice și de la limitele rețelelor complet conectate. Sunt analizate conexiunile locale, partajarea ponderilor, operația de convoluție, parametrii acesteia, funcțiile de activare, operația de pooling și rolul straturilor complet conectate. În final sunt discutate câteva arhitecturi reprezentative și modul în care acestea au contribuit la evoluția viziunii artificiale.

7.1. Reprezentarea caracteristicilor și operația de convoluție

În capitolul anterior au fost prezentate rețelele neuronale artificiale și modul în care acestea pot învăța o funcție de aproximare pornind de la exemple. O rețea de tip MLP poate primi ca intrare un vector de caracteristici și poate realiza clasificări sau regresii complexe. În cazul imaginilor apare însă o dificultate importantă: imaginea are o structură spațială care nu ar trebui ignorată.

O imagine digitală este reprezentată printr-o matrice de pixeli, așa cum arată Figura 7.1. Pentru o imagine RGB, fiecărui pixel îi sunt asociate trei valori, corespunzătoare canalelor de culoare. Astfel, o imagine de 1080×1080 pixeli are 1.166.400 de valori numerice. Din punct de vedere matematic, această imagine poate fi transformată într-un vector, însă o astfel de transformare pierde informația referitoare la vecinătatea pixelilor.

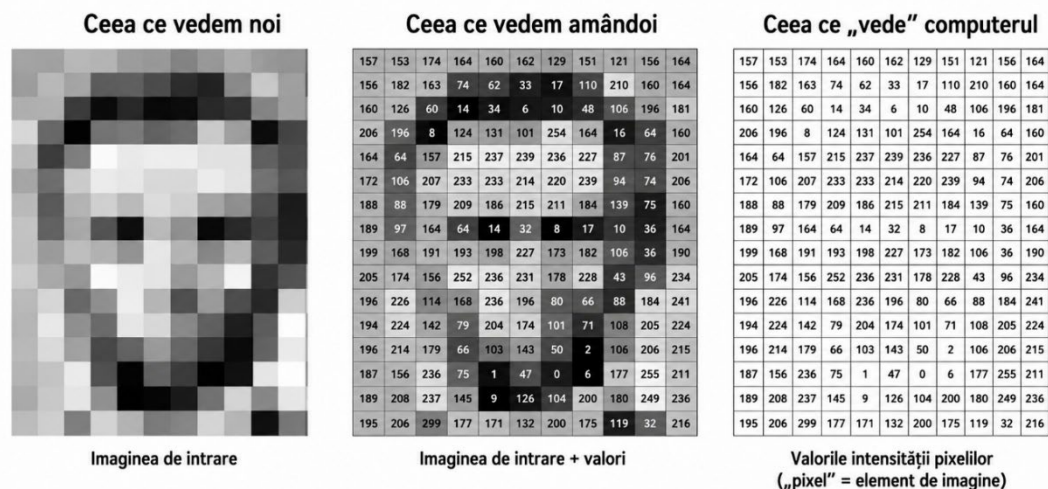


Fig. 7.1. Descompunerea unei imagini în pixeli

Aplatizarea imaginii într-un vector poate permite utilizarea unei rețele complet conectate, dar această abordare presupune că toate valorile de intrare sunt tratate în mod similar. Pentru imaginile naturale, acest lucru este problematic: pixelii aflați unul lângă altul sunt, de regulă, mai strâns legați din punct de vedere semantic decât pixelii aflați la distanțe mari.

Problema devine și mai evidentă atunci când dimensiunea imaginilor crește. Dacă o imagine are 96×96 pixeli, numai componenta spațială conține 9.216 valori pentru o imagine monocromă și 27.648 valori pentru una RGB. Dacă aceste valori sunt conectate complet la un strat ascuns cu 100 de neuroni, numărul ponderilor devine foarte mare. În plus față de costul de calcul, un număr mare de parametri crește și riscul de supra-învățare (*overfitting*).

Ideea centrală a rețelor convoluționale este restricționarea conexiunilor dintre niveluri consecutive. Un neuron nu mai primește informație de la întreaga imagine, ci de la o regiune locală. Regiunea respectivă este denumită câmp receptiv local (sau *patch*). În acest mod, arhitectura rețelei reflectă structura spațială a datelor.

Această idee este inspirată, la nivel conceptual, de organizarea cortexului vizual biologic, în care unele celule răspund preferențial la stimuli proveniți din anumite regiuni ale câmpului vizual. Într-o rețea CNN, însă, conexiunile locale sunt definite matematic și sunt învățate prin optimizarea parametrilor modelului. Avantajul major este că aceeași

caracteristică poate fi căutată în mai multe poziții ale imaginii. De exemplu, dacă un filtru învață să detecteze o margine verticală, filtrul poate fi aplicat pe întreaga imagine. Această proprietate conduce la două concepte fundamentale: conexiuni locale și partajarea ponderilor.

Într-o rețea convoluțională, scopul primelor straturi este de a transforma valorile brute ale pixelilor în reprezentări din ce în ce mai relevante. În loc să fie necesară definirea manuală a tuturor caracteristicilor, filtrele convoluționale sunt învățate automat în timpul antrenării. În cazul unei imagini monocrome, intrarea poate fi reprezentată ca o matrice cu două dimensiuni. Pentru o imagine color sunt necesare trei canale, iar pentru date video sau alte tipuri de date pot exista și mai multe canale. Prin urmare, intrarea unei CNN este tratată în general ca un volum de dimensiune înălțime $\times$ lățime $\times$ canale.

Un filtru convoluțional este o matrice de coeficienți care se deplasează peste imagine [3]. La fiecare poziție se calculează o combinație ponderată a valorilor din regiunea acoperită de filtru. Rezultatul este o nouă matrice, denumită hartă de caracteristici (*feature map*). Valorile mari din această hartă indică pozițiile în care caracteristica descrisă de filtru este prezentă mai puternic. Filtrele pot învăța să detecteze muchii, colțuri, tranziții de intensitate sau alte structuri locale [4]. În primele straturi, caracteristicile au de regulă o interpretare vizuală simplă. Pe măsură ce informația este transmisă către straturi mai profunde, caracteristicile pot deveni combinații mai complexe, asociate unor forme sau componente ale obiectelor.

Operația de convoluție utilizată în RNC poate fi privită ca o extensie a operației de combinație liniară întâlnită la neuronul artificial. Diferența importantă constă în faptul că aceeași matrice de ponderi este reutilizată în diferite poziții ale imaginii. Astfel, numărul parametrilor este mult mai mic decât în cazul unei conectări complete.



Fig. 7.2. Legătura dintre neuron și operația de convoluție

Pentru un filtru de dimensiune $q \times q$ și o regiune locală corespunzătoare, rezultatul unei poziții poate fi exprimat conceptual prin suma produselor dintre valorile filtrului și valorile imaginii, urmată, de regulă, de adăugarea unui termen de bias și aplicarea unei funcții de activare.

$$z(i, j) = \sum_a \sum_b K(a, b) * X(i + a, j + b) + b \quad (35)$$

Într-o RNC reală, filtrele operează pe toate canalele relevante ale intrării. Pentru o imagine RGB, un filtru poate avea dimensiunea $q \times q \times 3$. Mai multe filtre produc mai multe hărți de caracteristici, fiecare dintre acestea reprezentând răspunsul rețelei la un anumit tip de structură învățată.

Un aspect esențial este că filtrele nu sunt definite manual în timpul antrenării. Valorile coeficienților lor sunt parametri ai rețelei și sunt ajustate prin algoritmul de *backpropagation*, astfel încât activările filtrelor să fie utile pentru sarcina finală. Prin urmare, CNN-ul realizează simultan extragerea caracteristicilor și învățarea clasificatorului.

7.2. Parametrii operației de convoluție

Dimensiunea rezultatului unui strat convoluțional este determinată de mai mulți parametri. Dintre aceștia, cei mai importanți sunt dimensiunea filtrului, pasul de deplasare (*stride*) și padding-ul. Alegerea lor influențează atât dimensiunea reprezentărilor interne, cât și costul computațional al rețelei.

În absența padding-ului, filtrul poate fi poziționat doar acolo unde este cuprins în întregime în imagine. Din acest motiv, dimensiunea hărții de caracteristici este mai mică decât dimensiunea intrării. Repetarea operației în mai multe straturi poate conduce la o reducere semnificativă a dimensiunii spațiale.

Padding-ul constă în adăugarea unor valori suplimentare în jurul imaginii înaintea aplicării convoluției. Cel mai frecvent sunt utilizate valori zero. Prin această operație se poate controla reducerea dimensiunii spațiale și se poate evita pierderea rapidă a

informației de la marginea imaginii. În practică se întâlnesc frecvent două strategii. Padding-ul de tip valid nu adaugă valori suplimentare, astfel încât rezultatul este mai mic decât intrarea. Padding-ul de tip same urmărește păstrarea dimensiunii spațiale, atunci când combinația dintre dimensiunea filtrului și pasului de deplasare permite acest lucru.

Figura 7.3 ilustrează aplicarea padding-ului într-o operație de convoluție. Înaintea aplicării filtrului 3 x 3, matricea inițială este extinsă prin adăugarea unui strat de zerouri în jurul marginilor, astfel încât dimensiunea să crească de la 6 x 6 la 8 x 8. Această extindere permite filtrului să fie aplicat și asupra pixelilor aflați la marginea imaginii, fără a pierde informație din zonele de frontieră. În exemplu, padding-ul permite obținerea unei hărți de caracteristici cu dimensiunea 6 x 6, păstrând astfel dimensiunea spațială a imaginii inițiale.

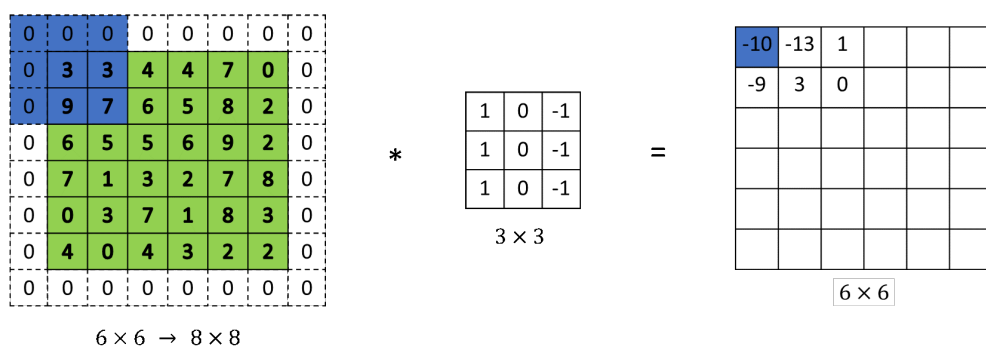


Fig. 7.3. Aplicarea padding-ului

Pasul de deplasare (stride) reprezintă numărul de poziții cu care este deplasată fereastra de convoluție la fiecare operație. Pentru stride = 1, filtrul este deplasat câte un pixel. Pentru stride = 2, sunt omise pozițiile intermediare, iar dimensiunea spațială a rezultatului se reduce mai rapid.

În forma generală, pentru o dimensiune spațială de intrare s_{zi} , filtru de dimensiune f , padding p și stride s , dimensiunea spațială a ieșirii poate fi exprimată prin:

$$s_{ieșire} = \text{floor} \left(\frac{s_{zi} - f + 2p}{s} \right) + 1 \quad (36)$$

Formula evidențiază faptul că arhitectura CNN poate controla explicit evoluția dimensiunii reprezentărilor interne. O alegere necorespunzătoare a parametrilor poate conduce la reprezentări prea mici sau la un cost de calcul inutil de mare.

Figura 7.4 ilustrează modul în care filtrul de dimensiune 3×3 glisează peste o matrice de intrare de 7×7 , arătând că la un pas (stride) de 1 filtrul se deplasează succesiv cu câte o singură poziție (pixel cu pixel), în timp ce la un pas (stride) de 2 acesta sare peste câte o poziție, parcurgând spațiul de intrare de două ori mai rapid și reducând dimensiunea caracteristicilor extrase.

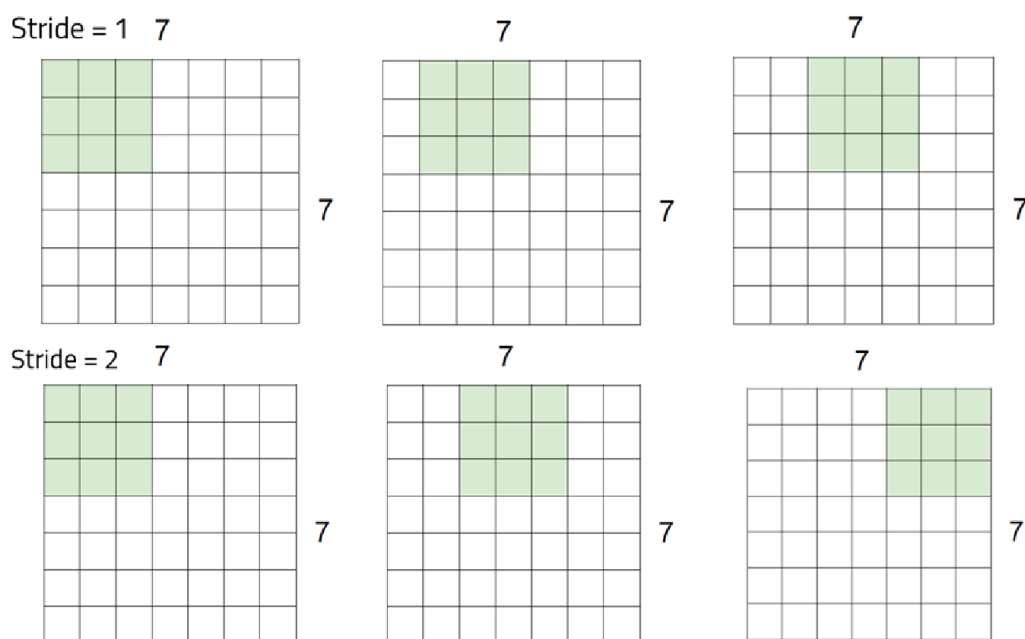


Fig. 7.4. Comparație între $\text{stride} = 1$ și $\text{stride} = 2$

După operația de convoluție este introdusă, în mod obișnuit, o funcție de activare neliniară. Una dintre cele mai utilizate este ReLU (Rectified Linear Unit), definită prin $\max(0, z)$. Ea păstrează valorile pozitive și înlocuiește valorile negative cu zero. Introducerea neliniarității permite rețelei să învețe reprezentări mult mai complexe decât cele obținute printr-o succesiune de transformări liniare.

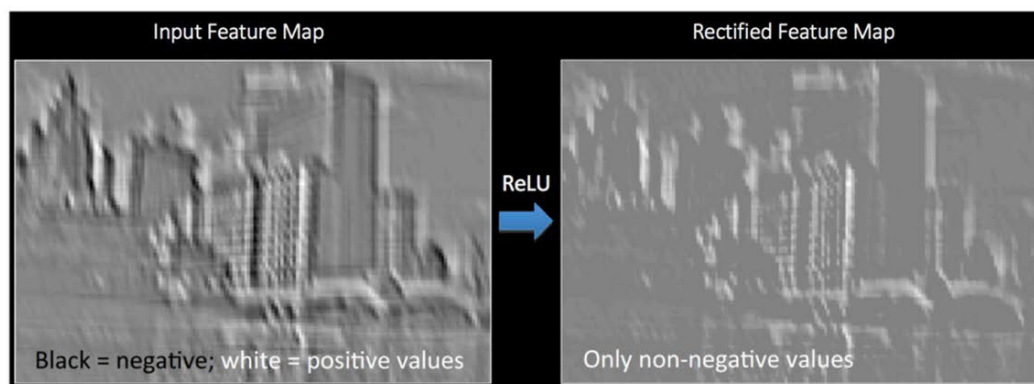


Fig. 7.5. *Aplicarea funcției de activare ReLU după operația de convoluție*

Figura 7.5 ilustrează efectul aplicării funcției de activare ReLU (Rectified Linear Unit) asupra unei hărți de caracteristici obținute în urma convoluției:

- **Input Feature Map (stânga):** Harta de caracteristici inițială conține atât valori pozitive, cât și negative; valorile negative sunt reprezentate prin nuanțe închise/negru, iar cele pozitive prin nuanțe deschise/alb.
- **Rectified Feature Map (dreapta):** După aplicarea funcției ReLU, toate valorile negative sunt eliminate (setate la 0), păstrându-se exclusiv valorile non-negative (reprezentate prin tonuri de la gri neutru la alb) pentru a introduce non-liniaritate în rețeaua neuronală.

7.3. Pooling și reducerea dimensiunii datelor

Pe măsură ce o rețea convoluțională procesează imaginea, numărul și complexitatea caracteristicilor pot crește. Pentru a controla dimensiunea reprezentărilor și costul computațional, RNC utilizează frecvent straturi de pooling.

Pooling-ul realizează o agregare locală a valorilor dintr-o hartă de caracteristici. În loc să păstreze fiecare valoare, o regiune este înlocuită cu o singură valoare reprezentativă. Cele mai cunoscute variante sunt max pooling și average pooling.

În cazul max pooling, pentru fiecare regiune este păstrată valoarea maximă. Aceasta poate fi interpretată ca păstrarea celei mai puternice activări al filtrului în regiunea

respectivă. În cazul average pooling, valorile sunt mediate, rezultând o reprezentare mai netedă.

Figura 7.6 exemplifică reducerea dimensiunii spațiale a unei matrice de 4 x 4 la una de 2 x 2 prin două metode de subeșantionare (pooling) aplicate pe ferestre nesuprapuse de 2 x 2:

- Mean pooling (stânga): Calculează valoarea medie din fiecare regiune (de exemplu, media valorilor galbenă 1, 3, 1, 3 este 2).
- Max pooling (dreapta): Extrage valoarea maximă din fiecare regiune (de exemplu, maximumul din regiunea galbenă 1, 3, 1, 3 este 3).

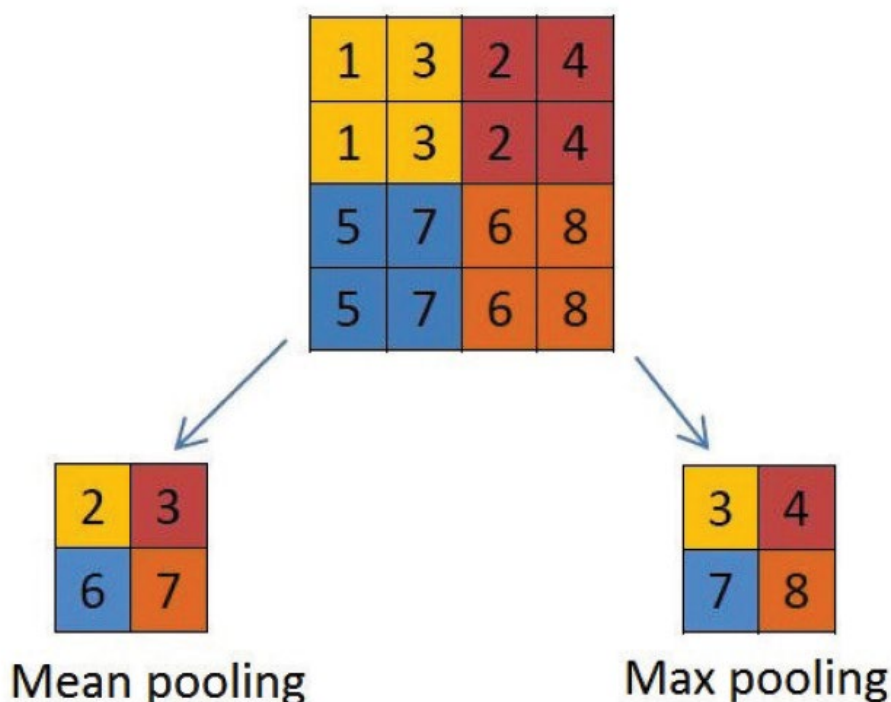


Fig. 7.6. Exemplu de reducere a dimensiunii prin pooling

Pooling-ul are mai multe efecte. În primul rând, reduce dimensiunea spațială a hărților de caracteristici și, implicit, costul operațiilor ulterioare. În al doilea rând, poate contribui la reducerea sensibilității la mici deplasări ale obiectului în imagine. De

asemenea, reducerea numărului de valori transmise către straturile finale poate limita riscul de *overfitting*.

Un aspect important este că pooling-ul clasic nu are parametri care trebuie învățați. Spre deosebire de un strat convoluțional, el aplică o regulă fixă de agregare. Operația se aplică separat pe fiecare canal al reprezentării, astfel încât numărul canalelor este păstrat, în timp ce dimensiunea spațială este redusă.

De exemplu, considerăm o imagine de 96×96 pixeli și un set de 400 de caracteristici extrase cu filtre de 8×8 . Fără reducerea dimensiunii, fiecare caracteristică poate genera o hartă spațială de dimensiune 89×89 , rezultând un număr foarte mare de valori. O astfel de reprezentare ar face costisitoare conectarea la un clasificator complet conectat și ar crește riscul de supra-învățare.

În arhitecturile moderne, pooling-ul nu este obligatoriu în forma sa clasică. Reducerea dimensiunii poate fi obținută și prin convoluții cu stride mai mare decât 1. Alegerea între pooling și convoluții cu stride depinde de arhitectura modelului și de proprietățile datelor.

7.4. Arhitectura unei Rețele Neuronale Convoluționale

O rețea CNN este alcătuită dintr-o succesiune de straturi care transformă progresiv imaginea inițială într-o reprezentare utilă pentru sarcina finală. Într-o arhitectură clasică, straturile convoluționale extrag caracteristici, straturile de pooling reduc dimensiunea spațială, iar straturile complet conectate realizează clasificarea.

În primele straturi, filtrele învață caracteristici cu nivel semantic redus, cum ar fi muchii și colțuri. În straturile intermediare, combinațiile acestor caracteristici pot descrie texturi, forme și părți ale obiectelor. În straturile profunde, reprezentările pot deveni suficient de abstracte pentru a caracteriza obiecte sau categorii.

Această organizare ierarhică reprezintă una dintre principalele diferențe dintre o RNC și o rețea complet conectată aplicată direct pe imagine. Într-o rețea MLP, transformarea imaginii în vector poate elimina relațiile spațiale. Într-o CNN, acestea sunt păstrate și exploatate prin conexiuni locale și partajarea ponderilor.

Dacă intrarea unui strat are dimensiunea $m \times n \times r$, unde m reprezintă înălțimea, n lățimea, iar r numărul de canale, un set de k filtre produce k hărți de caracteristici. Fiecare

filtru are propria matrice de ponderi și produce un singur canal de ieșire. Astfel, numărul de filtre controlează numărul canalelor reprezentării rezultate.

Stratul complet conectat are o funcționare similară cu cea întâlnită în rețelele neuronale clasice: fiecare neuron este conectat la toți neuronii stratului următor. În arhitecturile tradiționale, acest strat este utilizat în partea finală pentru clasificare. Dezavantajul său este numărul mare de parametri, motiv pentru care este preferabil ca reprezentarea să fie redusă înainte de acest strat. Figura 7.7 ilustrează arhitectura unui strat complet conectat specific etapei finale dintr-un CNN, în care fiecare neuron din stratul superior este conectat direct prin săgeți la toți neuronii din stratul inferior. Acest strat preia caracteristicile aplatizate, extrase anterior de straturile de convoluție și pooling, combinându-le global pentru a realiza decizia finală de clasificare.

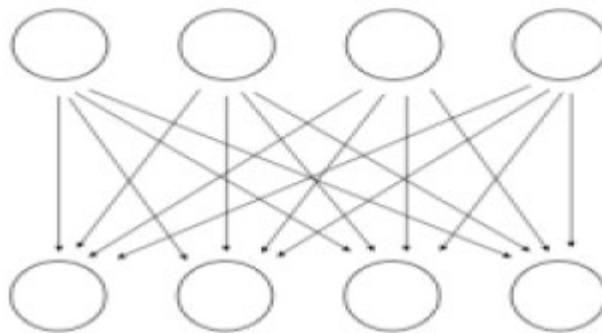


Fig. 7.7. Rolul stratului complet conectat în partea finală a unei CNN

În arhitecturile moderne, straturile complet conectate pot fi reduse sau eliminate. Clasificarea poate fi realizată prin operații de agregare și straturi convoluționale, ceea ce permite păstrarea mai mult timp a structurii spațiale și reducerea numărului de parametri.

O arhitectură CNN tipică poate fi descrisă schematic astfel:

$$[(CONV - ReLU) \times N - POOL?] \times M - (FC - ReLU) \times K - SOFTMAX$$

Figura 7.8 prezintă fluxul complet al unei arhitecturi CNN tipice. Aceasta evidențiază repetarea blocurilor convoluționale, urmate eventual de pooling, și utilizarea unor straturi complet conectate înaintea clasificării finale. În arhitecturile mai recente, numărul straturilor poate fi foarte mare, iar blocurile convoluționale pot include mecanisme suplimentare pentru optimizarea antrenării.

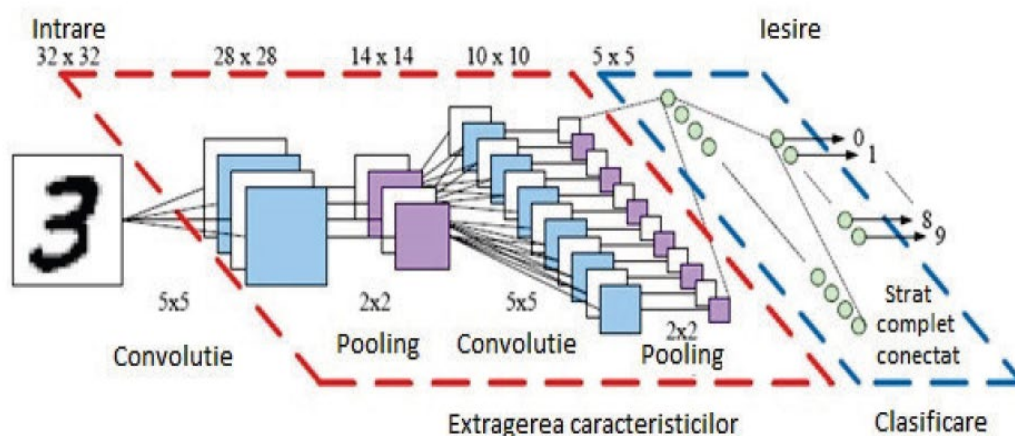


Fig. 7.8. Arhitectură CNN tipică

Astfel, diferența fundamentală dintre o rețea feed-forward complet conectată și o CNN constă în modul în care sunt tratate relațiile dintre elementele de intrare, așa cum ilustrează Figura 7.9.

Într-o rețea complet conectată, fiecare neuron poate primi informație de la toate valorile stratului precedent. Într-o CNN, conexiunile sunt locale, iar aceleași ponderi sunt reutilizate în mai multe poziții.

Această diferență are consecințe importante. CNN-urile necesită, în general, mai puțini parametri pentru aceeași problemă, păstrează informația spațială și pot identifica aceeași caracteristică indiferent de poziția sa în imagine. În schimb, arhitecturile complet conectate pot fi mai generale pentru date care nu au o structură spațială evidentă.

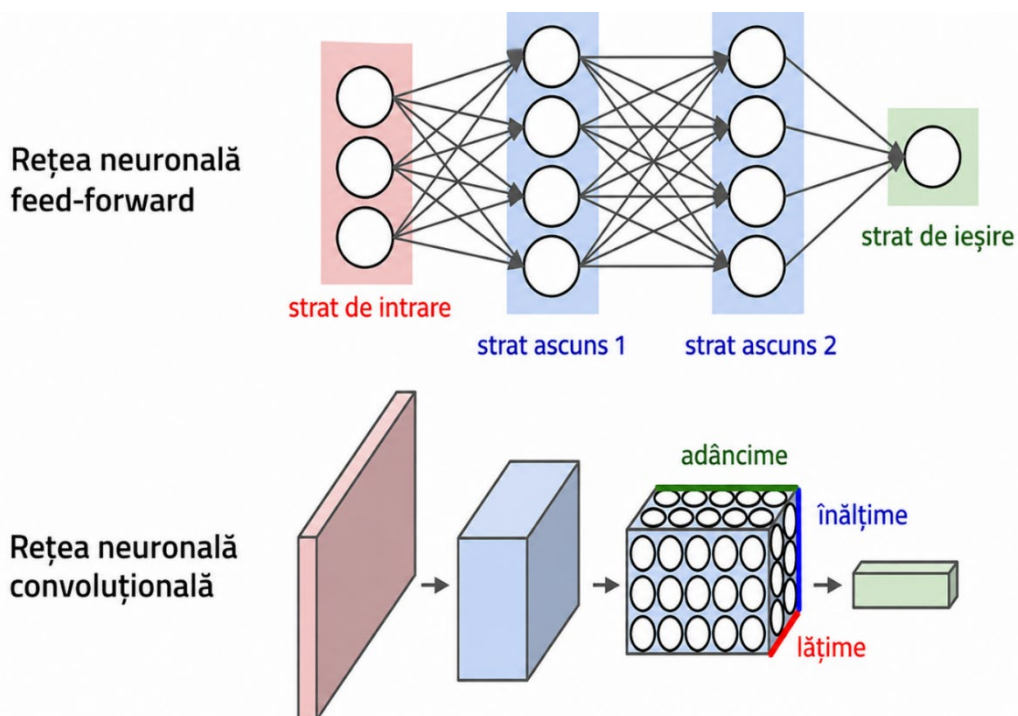


Fig. 7.9. Comparație între o rețea neuronală feed-forward și o rețea neuronală convoluțională

Într-o CNN, procesarea poate fi privită ca o succesiune de transformări. Imaginea este convertită mai întâi în hărți de caracteristici locale, apoi aceste caracteristici sunt combinate și agregate, iar în etapele finale sunt utilizate pentru determinarea clasei. Astfel, rețeaua învață automat o ierarhie de reprezentări.

7.5. Studii de caz : Evoluția Arhitecturilor CNN

Dezvoltarea rețelelor convoluționale poate fi urmărită prin câteva arhitecturi reprezentative, fiecare evidențiind o etapă importantă în evoluția procesării automate a imaginilor. LeNet-5 a demonstrat utilitatea convoluțiilor și a subsampling-ului pentru recunoașterea caracterelor, AlexNet a evidențiat potențialul rețelelor profunde antrenate pe volume mari de date, ResNet a introdus conexiunile reziduale pentru facilitarea antrenării arhitecturilor foarte adânci, iar autoencoderul a arătat cum componentele convoluționale pot fi utilizate pentru învățarea unor reprezentări compacte și pentru reconstrucția datelor.

Privite împreună, aceste exemple permit înțelegerea unor principii care au rămas esențiale în proiectarea RNC: utilizarea convoluțiilor succesive pentru extragerea caracteristicilor, reducerea progresivă a dimensiunilor spațiale, creșterea numărului și complexității caracteristicilor în straturile profunde, utilizarea unor mecanisme care facilitează antrenarea și, în funcție de obiectiv, transformarea reprezentării obținute într-o decizie de clasificare sau într-o reconstrucție a datelor.

Studiile de caz nu trebuie privite doar ca o succesiune de arhitecturi istorice. Ele constituie și exemple utile pentru înțelegerea modului în care se construiește o RNC și a motivelor pentru care anumite componente arhitecturale au devenit importante.

7.5.1. LeNet-5

LeNet-5 este una dintre arhitecturile clasice care au demonstrat eficiența convoluțiilor și a subsampling-ului pentru recunoașterea imaginilor. Arhitectura poate fi rezumată prin succesiunea CONV–POOL–CONV–POOL–CONV–FC. Filtrele convoluționale au dimensiunea 5×5 și sunt aplicate cu stride 1, iar operațiile de pooling utilizează regiuni 2×2 cu stride 2.

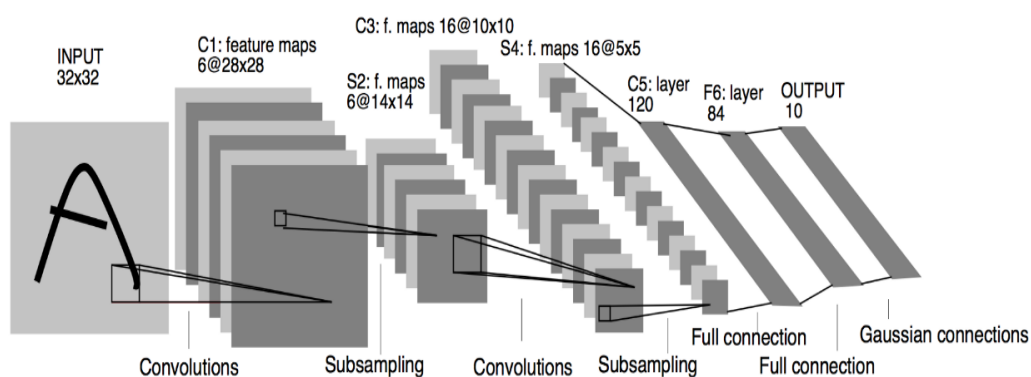


Fig. 7.10. *Arhitectura LeNet-5*

Un avantaj important al acestei arhitecturi este simplitatea succesiunii de transformări. Așa cum arată Fig. 7.10, imaginea de intrare nu este conectată direct la fiecare neuron din partea finală a rețelei. În schimb, informația este prelucrată local prin

filtre, iar rezultatele sunt organizate sub forma unor hărți de caracteristici. Această organizare reduce numărul de conexiuni și permite reutilizarea aceluiași ponderi în mai multe poziții ale imaginii.

Primele straturi convoluționale sunt orientate către identificarea unor structuri vizuale locale. Un filtru poate răspunde, de exemplu, la o anumită orientare a unei tranziții de intensitate. Valorile ponderilor sunt determinate în procesul de antrenare, astfel încât filtrele utile pentru sarcina urmărită să beneficieze de optimizare.

Operațiile de pooling reduc apoi dimensiunea spațială a reprezentărilor. În acest fel, straturile următoare lucrează cu un număr mai mic de poziții spațiale și pot integra informații provenite din regiuni mai largi ale imaginii. Reducerea rezoluției contribuie și la diminuarea costului de calcul.

Importanța arhitecturii constă nu numai în performanța sa pentru recunoașterea caracterelor, ci și în faptul că ilustrează foarte clar principiul ierarhiei de caracteristici. Primele convoluții extrag structuri simple, iar straturile următoare combină aceste informații pentru a obține reprezentări mai complexe.

Această ierarhie poate fi interpretată ca o trecere de la informație locală la informație din ce în ce mai abstractă. Într-o imagine care conține un caracter, de exemplu, o combinație de structuri locale poate contribui la identificarea unei părți a caracterului, iar mai multe astfel de părți pot contribui la recunoașterea întregului. Partea finală a rețelei are rolul de a transforma caracteristicile extrase într-o decizie de clasificare. După reducerea și combinarea informației în straturile convoluționale, reprezentarea poate fi transmisă către straturi complet conectate, care realizează asocierea dintre caracteristicile învățate și clasele urmărite.

LeNet-5 este, prin urmare, un exemplu foarte bun pentru înțelegerea fluxului general dintr-o RNC: imaginea este transformată succesiv, caracteristicile sunt extrase și combinate, dimensiunea spațială este redusă, iar reprezentarea finală este utilizată pentru clasificare.

7.5.2. AlexNet

AlexNet a reprezentat un moment important în evoluția învățării profunde pentru clasificarea imaginilor. Această arhitectură are o imagine de intrare de $227 \times 227 \times 3$ și primul strat convoluțional cu 96 de filtre de 11×11 , aplicate cu stride 4.

Dimensiunea ieșirii poate fi calculată astfel: $(227 - 11) / 4 + 1 = 55$. Rezultă o reprezentare spațială de 55×55 , cu 96 de hărți de caracteristici.

Numărul parametrilor în primul strat poate fi calculat prin înmulțirea numărului de valori dintr-un filtru cu numărul de filtre: $(11 \times 11 \times 3) \times 96 = 34.848$. Dacă se consideră și termenii bias, se adaugă câte un parametru pentru fiecare filtru. Acest lucru evidențiază faptul că, într-o CNN, numărul parametrilor este determinat de dimensiunea filtrelor și de numărul canalelor de intrare și de ieșire.

Comparativ cu LeNet-5, AlexNet utilizează o arhitectură semnificativ mai mare și mai profundă (Figura 7.11). Creșterea numărului de straturi și de filtre conduce la o capacitate mai mare de a reprezenta structuri vizuale complexe. În același timp, numărul parametrilor și costul de antrenare cresc.

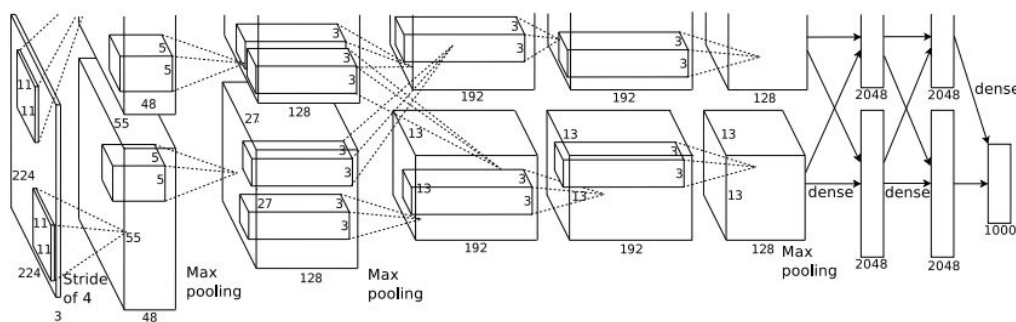


Fig. 7.11. Arhitectura AlexNet

Un element important al arhitecturii AlexNet este utilizarea funcției de activare ReLU. Introducerea unei funcții de activare neliniare este esențială deoarece, fără neliniaritate, succesiunea mai multor transformări liniare ar putea fi echivalentă cu o singură transformare liniară. ReLU permite rețelei să construiască reprezentări neliniare și contribuie la eficiența procesului de optimizare.

AlexNet este și un exemplu important pentru înțelegerea relației dintre arhitectură și resursele de calcul. O rețea cu multe convoluții și un număr mare de parametri necesită un volum important de calcule în timpul antrenării. Procesarea paralelă a contribuit la posibilitatea antrenării unor astfel de modele într-un timp rezonabil.

Pe lângă capacitatea mare de reprezentare, o rețea cu mulți parametri este mai expusă riscului de supra-învățare. Din acest motiv, în proiectarea și antrenarea modelelor de acest tip sunt importante tehnicile de regularizare, augmentarea datelor și alegerea corespunzătoare a hiperparametrilor.

AlexNet a consolidat ideea că extragerea caracteristicilor poate fi realizată automat, direct din date. Prin urmare, poate fi considerat un punct de referință pentru trecerea către modele convoluționale profunde, în care creșterea capacității de reprezentare este susținută de date, algoritmi de optimizare și resurse de calcul adecvate.

7.5.3. ResNet

Pe măsură ce rețelele au devenit mai profunde, antrenarea lor a devenit mai dificilă. ResNet a introdus conexiuni de tip *shortcut* (sau *skip connection*), prin care informația poate ocoli anumite straturi și poate fi transmisă direct către o etapă ulterioară. Această idee facilitează propagarea gradientului și permite construirea unor rețele cu adâncime mult mai mare.

Din punct de vedere conceptual, conexiunea shortcut nu elimină transformarea realizată de blocul convoluțional, ci îi oferă rețelei o cale suplimentară pentru transmiterea informației. Modelul poate învăța astfel o transformare reziduală, ceea ce simplifică optimizarea în cazul arhitecturilor profunde, așa cum ilustrează Figura 7.12.

Un bloc rezidual poate fi descris simplificat prin relația $y = F(x) + x$, unde x reprezintă intrarea blocului, iar $F(x)$ reprezintă transformarea realizată de straturile din interior. Termenul x este transmis pe conexiunea shortcut și este apoi combinat cu rezultatul transformării.

În loc să construiască direct o transformare completă $H(x)$, partea convoluțională poate învăța funcția reziduală $F(x) = H(x) - x$. Dacă identitatea este o aproximare bună, blocul nu trebuie să învețe o transformare complexă, ci poate păstra informația existentă și să învețe doar modificarea necesară. Conexiunea shortcut este importantă și pentru propagarea gradientului. Într-o rețea foarte adâncă, gradientul trebuie transmis prin

numeroase transformări succesive. Călea directă oferită de shortcut poate facilita transmiterea informației și a gradientului către straturile precedente.



Fig. 7.12. Principiul arhitecturii ResNet și utilizarea conexiunilor de tip shortcut

Dacă dimensiunile reprezentărilor de la intrarea și ieșirea blocului nu coincid, conexiunea directă poate include o transformare, de exemplu o convoluție, astfel încât cele două componente să poată fi adunate. Principiul rămâne însă același: păstrarea unei căi directe pentru informație.

ResNet arată că dezvoltarea unei rețele nu trebuie realizată doar prin adăugarea succesivă de convoluții. Este necesar și un mecanism arhitectural care să permită optimizarea eficientă a modelului. Conexiunile reziduale oferă un astfel de mecanism.

Conceptul de conexiune shortcut a devenit relevant și pentru alte arhitecturi de rețele neuronale. Ideea generală este că o rețea profundă poate beneficia de existența unor căi care permit informației să circule pe rute mai scurte, în timp ce transformările învățate de blocurile intermediare ajustează reprezentarea.

7.5.4. Autoencoder

Autoencoder-ul este o arhitectură neuronală care poate fi utilizată pentru învățarea unei reprezentări compacte a datelor. În cazul imaginilor, partea de encoder transformă imaginea într-o reprezentare latentă cu dimensiune redusă, iar partea de decoder încearcă să reconstruiască imaginea inițială. Într-o variantă convoluțională, operațiile de codificare și decodificare pot exploata structura spațială a imaginii.

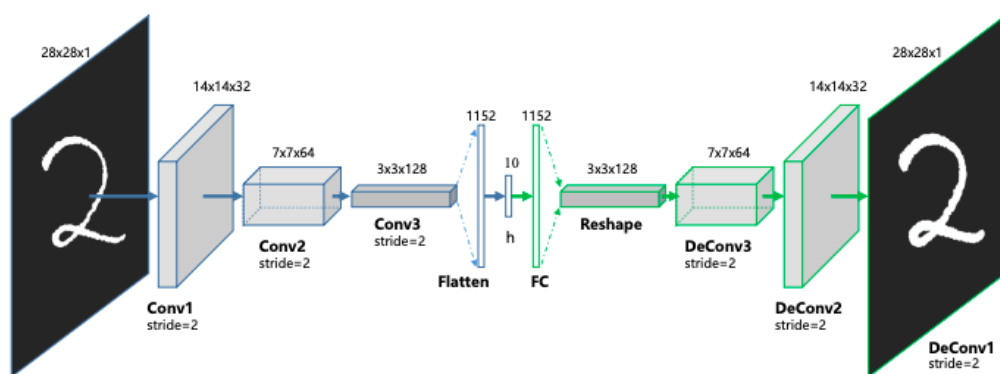


Fig. 7.13. Principiul unui autoencoder

Spre deosebire de un clasificator, autoencoderul nu are ca obiectiv principal atribuirea unei clase unei imagini. Obiectivul este ca reprezentarea latentă să păstreze suficientă informație pentru ca decoderul să poată produce o reconstrucție apropiată de intrarea inițială.

Encoderul poate conține convoluții, funcții de activare și operații de reducere a dimensiunii. Pe măsură ce informația avansează prin encoder, dimensiunea spațială poate scădea, în timp ce reprezentarea poate deveni mai bogată în canale. Rezultatul este o reprezentare compactă.

Decoderul efectuează procesul invers. Pornind de la reprezentarea latentă, acesta reconstruiește progresiv dimensiunea spațială. Pot fi utilizate operații de upsampling, convoluții sau convoluții transpuse. Scopul este obținerea unei imagini reconstruite cât mai apropiată de imaginea de intrare.

Dacă imaginea inițială este notată cu x , iar imaginea reconstruită cu $\hat{x}$, procesul de antrenare poate minimiza o funcție de pierdere de reconstrucție. O variantă simplă este eroarea pătratică medie, exprimată prin relația $L = \frac{1}{N} * \sum (x_i - \hat{x}_i)^2$.

Prin minimizarea acestei funcții, encoderul este determinat să păstreze în reprezentarea latentă informația relevantă pentru reconstrucție, iar decoderul învață să utilizeze această reprezentare pentru reconstrucția datelor.

Dimensiunea reprezentării latente este importantă. Dacă aceasta este foarte mare, modelul poate ajunge să păstreze aproape integral informația de intrare, iar efectul de comprimare este redus. Dacă este prea mică, o parte importantă a informației poate fi pierdută, iar reconstrucția poate deveni mai puțin precisă.

Autoencoderele convoluționale sunt potrivite pentru imagini deoarece encoderul exploatează aceleași principii locale ca o CNN de clasificare. Filtrele detectează structuri vizuale, iar transformările succesive produc o reprezentare din ce în ce mai compactă.

Una dintre utilizările importante este reducerea dimensionalității. Autoencoderul învață reprezentarea latentă prin optimizare pe baza datelor disponibile, astfel încât transformarea să fie adaptată structurii acestora.

O altă aplicație este detectarea anomalilor. Dacă modelul este antrenat predominant pe date considerate normale, acesta poate învăța structura caracteristică acestor exemple. Atunci când primește o observație care diferă semnificativ de tiparul învățat, reconstrucția poate fi mai slabă, ceea ce conduce la o eroare de reconstrucție

mai mare. Într-un astfel de sistem, eroarea de reconstrucție poate fi transformată într-un scor de anomalie. Pentru o valoare stabilită a pragului, observațiile cu eroare peste prag pot fi considerate potențial anormale. Alegerea pragului trebuie însă realizată în funcție de date și de obiectivul aplicației.

Autoencoderul poate fi utilizat și pentru *denoising*. În acest caz, intrarea furnizată encoderului poate fi o versiune perturbată a imaginii, iar ținta de reconstrucție este imaginea mai puțin afectată de zgomot. Modelul este astfel determinat să păstreze structurile relevante și să reducă informația nedorită.

Reprezentarea latentă poate fi analizată și independent de reconstrucție. Ea poate fi utilizată pentru vizualizarea datelor, grupare, clasificare ulterioară sau ca intrare pentru un alt model. În acest caz, autoencoderul poate funcționa ca un mecanism de învățare automată a caracteristicilor.

Este important ca o eroare de reconstrucție mare să nu fie interpretată automat ca dovadă a unei anomalii. Rezultatul depinde de datele de antrenare, de capacitatea modelului, de distribuția observațiilor și de alegerea pragului. Autoencoderul oferă un criteriu bazat pe reconstrucție, care trebuie evaluat în contextul aplicației.

Într-un autoencoder convoluțional, encoderul reduce treptat reprezentarea până la o formă compactă, iar decoderul încearcă să reconstruiască datele inițiale. Spre deosebire de un clasificator, obiectivul nu este neapărat atribuirea unei clase, ci învățarea unei reprezentări care păstrează informația relevantă pentru reconstrucție.

8. Capitolul 8. Prelucrarea limbajului natural. Arhitecturi transformer

Prelucrarea limbajului natural (*Natural Language Processing* – NLP) reprezintă domeniul în care metodele de inteligență artificială și învățare automată sunt utilizate pentru analiza, reprezentarea și generarea limbajului natural uman. În cazul textului, problema fundamentală este diferită de cea întâlnită în cazul datelor numerice sau al imaginilor: informația este transmisă prin cuvinte, ordine, context și relații lingvistice, iar aceeași formă lexicală poate avea interpretări diferite în funcție de context [11].

Capitolul urmărește evoluția de la tehnicile preliminare de prelucrare a textului și reprezentările clasice către reprezentări semantice dense și, în final, către arhitecturile de tip Transformer. Sunt discutate tokenizarea, segmentarea, normalizarea, eliminarea cuvintelor comune și lematizarea, urmate de reprezentări precum Bag-of-Words, TF-IDF, n-grame, matrici de co-ocurență și embeddinguri. În continuare sunt prezentate metode de clasificare a textului și principiile arhitecturilor Transformer, cu accent pe BERT, GPT și T5.

Un principiu important este acela că performanța unui sistem NLP depinde nu numai de algoritmul de învățare, ci și de modul în care textul este transformat într-o reprezentare numerică. Pipeline-ul de prelucrare trebuie proiectat astfel încât să păstreze informația relevantă pentru sarcina urmărită și să reducă elementele care introduc zgomot sau redundanță. În modelele moderne, această transformare este completată de reprezentări contextuale, care permit modelului să țină cont de relațiile dintre tokeni în cadrul întregii secvențe.

8.1. NLP și pre-procesarea textului

Prelucrarea limbajului natural începe, de regulă, cu text brut. Spre deosebire de un tabel numeric, textul nu poate fi introdus direct într-un algoritm clasic de învățare automată. Este necesară transformarea sa într-o formă procesabilă, printr-o serie de operații care urmăresc curățarea, uniformizarea și structurarea informației.

Aceste tehnici preliminare includ preprocesarea, tokenizarea, segmentarea și normalizarea, iar ulterior eliminarea cuvintelor comune, lematizarea și etichetarea gramaticală. Aceste operații nu trebuie considerate reguli absolute, alegerea lor depinzând de sarcină și de tipul textului analizat.

Preprocesarea textului reprezintă primul filtru prin care trece documentul. Obiectivele sale principale sunt îndepărtarea zgomotului, uniformizarea reprezentării, reducerea dimensiunii datelor și pregătirea pentru analiza semantică și sintactică ulterioară. În această etapă pot fi eliminate caractere redundante, semne de punctuație sau diferențe de capitalizare.

Eliminarea semnelor de punctuație poate fi utilă atunci când acestea nu aduc informație pentru sarcina urmărită. În schimb, pentru anumite aplicații, punctuația poate fi relevantă. De aceea, preprocesarea trebuie privită ca un set de operații configurabile și nu ca o secvență care trebuie aplicată identic tuturor textelor.

Conversia la litere mici reduce diferențele artificiale dintre formele cu majuscule și minuscule. De exemplu, „Marele Ocean Pacific” poate deveni „marele ocean pacific”. Operația reduce numărul de forme distincte care trebuie reprezentate în vocabular și poate simplifica procesarea ulterioară. Totuși, capitalizarea poate transmite informații în anumite sarcini, de exemplu în recunoașterea entităților, motiv pentru care această operație trebuie aleasă în funcție de obiectiv.

Numerele și caracterele speciale sunt, de asemenea, tratate diferit în funcție de domeniu. În analiza unor texte narative, eliminarea numerelor poate reduce zgomotul. În schimb, în texte financiare sau tehnice, valorile numerice pot fi esențiale. În cazul textelor provenite din rețele sociale, emoji-urile și hashtag-urile pot conține informații importante despre sentiment și intenție. Astfel, aceste etape necesită o decizie dependentă de domeniu.

Figura 8.1 ilustrează succesiunea principalelor etape de prelucrare a textului înainte de aplicarea modelelor NLP: tokenizarea (prin împărțirea textului în unități mici) segmentarea (prin identificarea limitelor propozițiilor), și normalizarea, prin transformarea textului într-o formă uniformă.

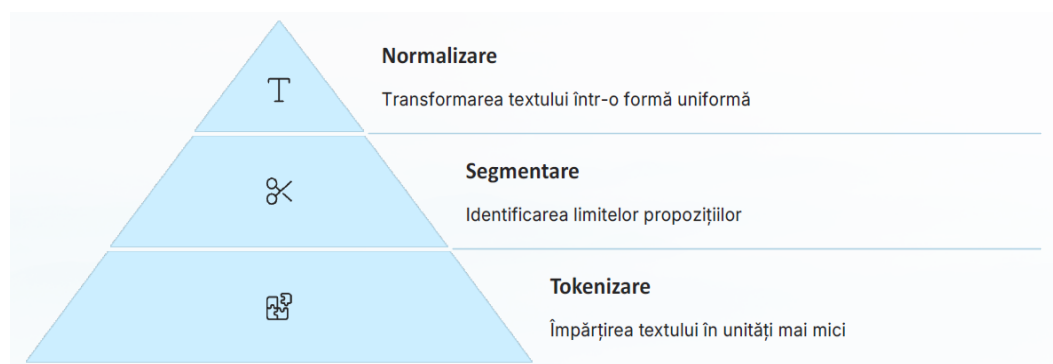


Fig. 8.1. Etapele preliminare ale procesării limbajului natural

Tokenizarea transformă textul într-o succesiune de unități numite tokeni. Aceste unități pot fi cuvinte, propoziții sau, în cazul modelelor moderne, fragmente de cuvinte. Calitatea tokenizării influențează direct etapele ulterioare deoarece toate operațiile de reprezentare și învățare se bazează pe secvența de tokeni rezultată.

Tokenizarea la nivel de cuvinte este o abordare intuitivă: propoziția este împărțită în cuvinte individuale. De exemplu, „Prelucrarea limbajului natural” poate deveni secvența [„Prelucrarea”, „limbajului”, „natural”]. Această reprezentare este ușor de înțeles, dar poate conduce la un vocabular mare, mai ales în limbi cu multe forme flexionate.

O altă operație este segmentarea propozițiilor. Textul este împărțit în unități propoziționale care pot fi procesate separat sau utilizate ca elemente ale unui context mai larg. Problema nu este însă trivială, deoarece punctul poate indica sfârșitul unei propoziții sau poate apărea într-o abreviere. Un sistem de segmentare trebuie să distingă aceste situații pentru a evita fragmentarea incorectă a contextului.

Normalizarea urmărește obținerea unei forme consistente a textului. Aceasta presupune conversia la litere mici, corectarea ortografică și expansiunea abrevierilor. De exemplu, „limbaj naturral” poate fi corectat la „limbaj natural”, iar o abreviere poate fi transformată în forma sa completă. Aceste operații reduc variația superficială și pot îmbunătăți calitatea reprezentării.

Lematizarea constă în reducerea unui cuvânt la forma sa de bază, numită leamă. Procesul implică analiza morfologică și ține cont de formele flexionate ale cuvintelor, așa cum ilustrează Figura 8.2. Într-un exemplu simplu, forme precum „mergeam” și „mergem” pot fi reduse la „merge” (de la verbul „a merge”). Pentru limba română, această operație este deosebit de utilă deoarece aceeași bază lexicală poate apărea în multe forme gramaticale [9].

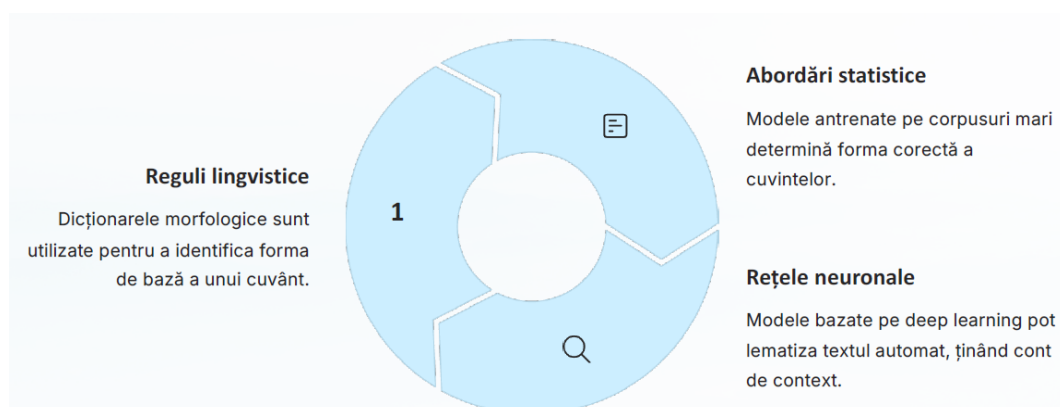


Fig. 8.2. Principiul lematizării

Eliminarea cuvintelor comune, denumite stopwords, reprezintă o altă tehnică de reducere a dimensiunii. În această categorie pot intra cuvinte precum „și”, „dar”, „de”, „sau”, „în”, „pe” și „la”. În multe aplicații, aceste cuvinte apar foarte frecvent și contribuie puțin la diferențierea semantică dintre documente. Eliminarea lor poate accelera procesarea și reduce dimensiunea reprezentării. □

Forma flexionată	Lemă (forma de bază)	Parte de vorbire
cântărețul	cântăreț	substantiv
alegerile	alegere	substantiv
plecând	pleca	verb
copiii	copil	substantiv
mergeau	merge	verb
frumoasă	frumos	adjectiv

Fig. 8.3. Exemple de lematizare

Totuși, aceste stopwords nu sunt întotdeauna irelevante. În unele sarcini lingvistice, cuvintele gramaticale contribuie la relațiile sintactice sau la exprimarea intenției. În consecință, o listă de stopwords trebuie adaptată domeniului și sarcinii. Pot fi utilizate atât liste predefinite, disponibile în biblioteci precum NLTK și spaCy, cât și liste personalizate, adaptate domeniului și specificului aplicației. Alte exemple de lematizare sunt ilustrate în Figura 8.3.

Astfel, un pipeline NLP poate integra curățarea, tokenizarea, normalizarea, lematizarea și, în funcție de aplicație, etichetarea gramaticală sau recunoașterea entităților (Figura 8.4). Avantajul unui astfel de flux modular constă în faptul că etapele pot fi înlocuite sau adaptate fără reconstruirea întregii aplicații.



Fig. 8.4. Pipeline tipic pentru procesarea limbajului natural

8.2. Reprezentarea textului: de la Bag-of-words la embeddinguri

După preprocesare, textul trebuie transformat într-o reprezentare numerică. Modelele de învățare automată operează asupra valorilor numerice, astfel încât reprezentarea aleasă determină în mare măsură informația disponibilă algoritmului. În evoluția NLP pot fi observate mai multe etape: reprezentări bazate pe frecvența cuvintelor, reprezentări care introduc context local și, ulterior, embeddinguri dense și reprezentări contextuale, așa cum arată și Figura 8.5.

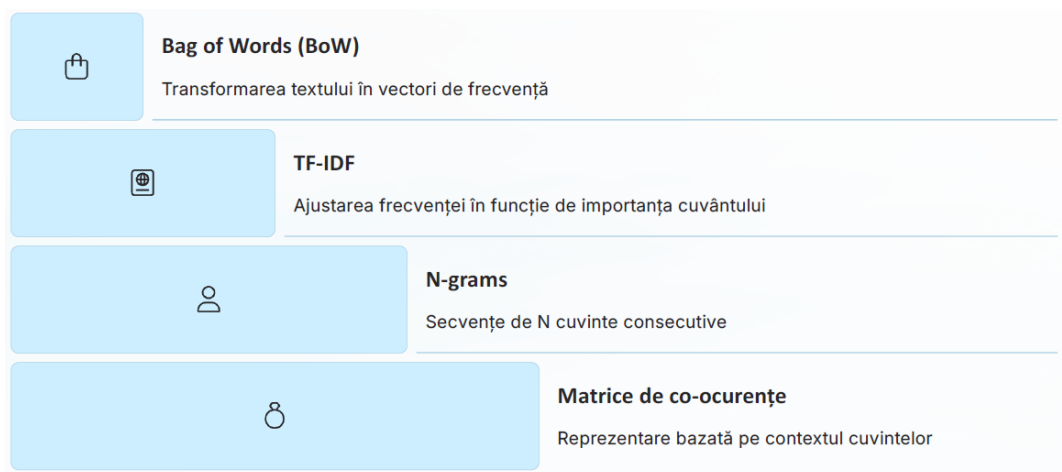


Fig. 8.5. Tehnici de preprocesare în NLP

Modelul Bag-of-Words construiește un vocabular alcătuit din cuvintele distincte ale corpusului și reprezintă fiecare document printr-un vector ale cărui componente corespund frecvențelor cuvintelor. Metoda este simplă, ușor de implementat și poate funcționa bine pentru clasificarea textului, însă ignoră ordinea cuvintelor și relațiile semantice dintre ele.

Această abordare are avantajul unei implementări simple și al unui cost computațional redus, ceea ce o face potrivită pentru numeroase modele clasice de clasificare a textului. Reprezentarea este însă caracterizată de o dimensionalitate ridicată, deoarece fiecare cuvânt din vocabular corespunde unei componente a vectorului, iar pentru corpusuri mari rezultă vectori foarte mari. De asemenea, nu păstrează ordinea cuvintelor și nu surprinde relațiile semantice sau contextul în care acestea apar,

astfel încât două propoziții cu structuri sau sensuri diferite pot primi reprezentări foarte asemănătoare.

Pentru un exemplu simplificat, presupunem vocabularul [„mare”, „ocean”, „frumos”]. Documentul „mare ocean” poate fi reprezentat prin [1,1,0], iar „ocean frumos” prin [0,1,1]. Vectorii descriu prezența sau frecvența termenilor, dar nu spun că în primul document „mare” și „ocean” apar într-o anumită relație sintactică.

TF-IDF îmbunătățește această idee prin diminuarea influenței cuvintelor care apar foarte frecvent în multe documente. Componenta TF măsoară frecvența termenului în document, iar componenta IDF (*Inverse Document Frequency*) penalizează termenii răspândiți în întregul corpus. IDF este definit prin logaritmul raportului dintre numărul total de documente și numărul documentelor care conțin termenul.

$$IDF(t) = \log\left(\frac{n}{N}\right) \quad (37)$$

TF-IDF este util în motoare de căutare și clasificare deoarece acordă o importanță mai mare termenilor specifici unui document și una mai mică termenilor foarte comuni. Totuși, reprezentarea rămâne una bazată pe frecvențe și nu modelează direct semnificația contextuală.

N-gramele introduc informație despre ordinea locală a cuvintelor. Un unigram este un singur cuvânt, un bigram este o pereche de cuvinte consecutive, iar un trigram conține trei cuvinte consecutive. Astfel, expresia „învățare automată” poate fi reprezentată printr-un bigram, oferind mai mult context decât două unigrame independente.

Avantajul n-gramelor este că păstrează o parte din structura secvențială și pot surprinde expresii frecvente. Dezavantajul este creșterea rapidă a numărului de combinații posibile și, implicit, a dimensionalității reprezentării. Pentru corpusuri mari, numărul de n-grame distincte poate crește foarte mult.

Matricea de co-ocurențe urmărește cât de frecvent apar cuvintele în vecinătatea altor cuvinte. Rândurile pot reprezenta termenii din vocabular, iar coloanele termenii care apar în context. O astfel de matrice poate fi foarte mare, motiv pentru care pot fi utilizate metode de reducere a dimensionalității, precum SVD (*Singular Value Decomposition*).

Pe această bază se ajunge la ideea de spațiu semantic. Într-o reprezentare vectorială, fiecare cuvânt este asociat unui vector, iar cuvintele care apar în contexte asemănătoare tind să fie reprezentate prin puncte apropiate. Astfel, distanța dintre vectori poate fi folosită ca indicator al asemănării semantice.

Un exemplu simplu de generare a embeddingurilor este *Word2Vec*. Biblioteca *gensim* poate fi utilizată pentru antrenarea unui model *Word2Vec* pe un set de două propoziții scurte. Codul exemplifică modul în care fiecare cuvânt este asociat unei reprezentări sub forma unui vector numeric. Cu un corpus atât de mic, însă, vectorii nu trebuie interpretați ca reprezentări semantice robuste.

```
from gensim.models import Word2Vec
sentences = [["Merg", "la", "mare"], ["Îmi", "place", "marea"]]
model = Word2Vec(sentences, vector_size=50, window=2,
min_count=1)
print(model.wv["mare"]) # Vectorul numeric al cuvântului "mare"
```

Fig. 8.6 ilustrează reprezentarea numerică embedding-ului sub forma unui vector dens de numere reale. Spre deosebire de reprezentările clasice de tip one-hot encoding, care sunt matrice rare compuse aproape exclusiv din zeros, acest vector comprimă semnificația semantică a unui cuvânt într-un spațiu vectorial continuu.

```
[ 1.56351421e-02 -1.90203730e-02 -4.11062239e-04  6.93839323e-03
 -1.87794445e-03  1.67635437e-02  1.80215668e-02  1.30730132e-02
 -1.42324204e-03  1.54208085e-02 -1.70686692e-02  6.41421322e-03
 -9.27599426e-03 -1.01779103e-02  7.17923651e-03  1.07406788e-02
  1.55390287e-02 -1.15330126e-02  1.48667218e-02  1.32509926e-02
 -7.41960062e-03 -1.74912829e-02  1.08749345e-02  1.30195115e-02
 -1.57510047e-03 -1.34197120e-02 -1.41718509e-02 -4.99412045e-03
  1.02865072e-02 -7.33047491e-03 -1.87401194e-02  7.65347946e-03
  9.76895820e-03 -1.28571270e-02  2.41711619e-03 -4.14975407e-03
  4.88066689e-05 -1.97670180e-02  5.38400887e-03 -9.50021297e-03
  2.17529293e-03 -3.15244915e-03  4.39334614e-03 -1.57631524e-02
 -5.43436781e-03  5.32639725e-03  1.06933638e-02 -4.78302967e-03
 -1.90201886e-02  9.01175756e-03]
```

Fig. 8.6. Reprezentarea vectorială densă a unui embedding

Figura 8.7 prezintă o vizualizare bidimensională a spațiului de embedding, obținută prin reducerea dimensionalității vectorilor densi de la un spațiu multidimensional la un plan 2D (de exemplu, prin tehnici precum PCA). Graficul demonstrează capacitatea embedding-urilor de a grupa automat cuvintele în funcție de similaritatea lor semantică. Astfel, se observă că noțiunile legate de baie și obiecte sanitare (bathroom, sink, toilet) formează un grup unitar în stânga, electrocasnicele (refrigerator, oven, microwave) sunt grupate în partea superioară, sculele și uneltele de lucru (drill, saw, bosch, dewalt) se concentrează în dreapta, iar termenii legați de grădinărit (hose, sprinkler, grass) se află în zona inferioară. Această reprezentare spațială confirmă că această metodă transpune proximitatea conceptuală și contextuală a cuvintelor în proximitate geometrică.

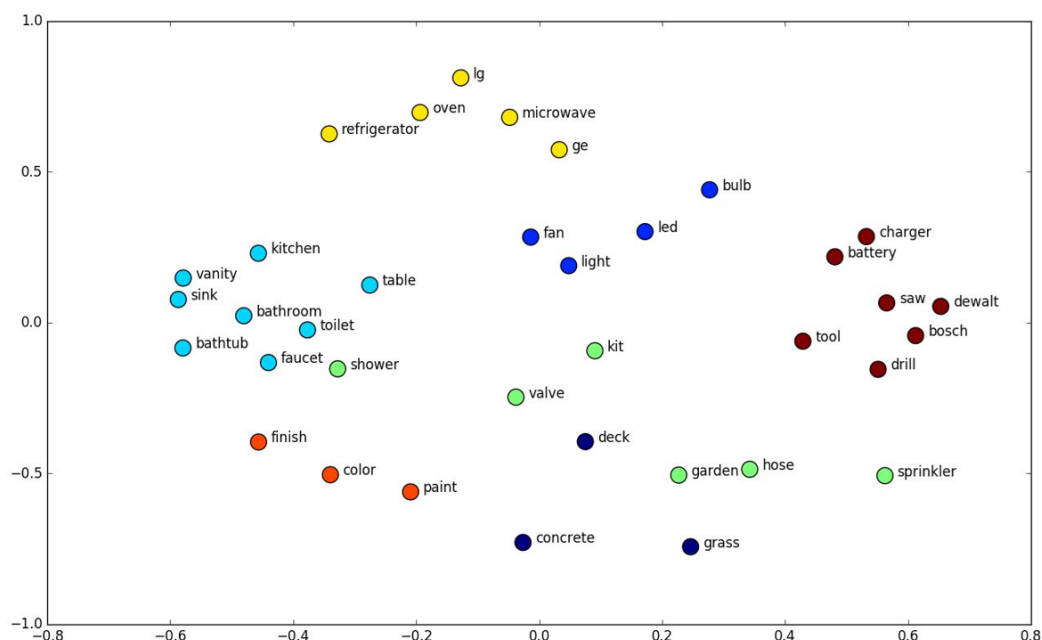


Fig. 8.7. Vizualizarea bidimensională a embeddingurilor

Odată obținute embeddingurile, similaritatea dintre doi termeni poate fi evaluată, de exemplu, prin similaritatea cosinus. Acesta ia, de regulă, o valoare între 0 și 1, cu valori mai mari indicând o apropiere semantică mai mare în spațiul vectorial.

$$\cos(\theta) = \frac{u * v}{||u|| ||v||} \quad (38)$$

Figura 8.8 prezintă sintetic evoluția și tipologia spațiilor de reprezentare vectorială a textului în procesarea limbajului natural (NLP). Diagrama compară trei abordări fundamentale: modelele clasice bazate pe frecvență (precum Bag-of-Words și TF-IDF), spațiile distribuționale care generează reprezentări dense statice (ex. Word2Vec, GloVe) și spațiile contextuale avansate (precum modelele bazate pe arhitectura Transformer, de exemplu BERT), care adaptează reprezentarea dinamic în funcție de contextul fiecărei apariții a cuvântului.

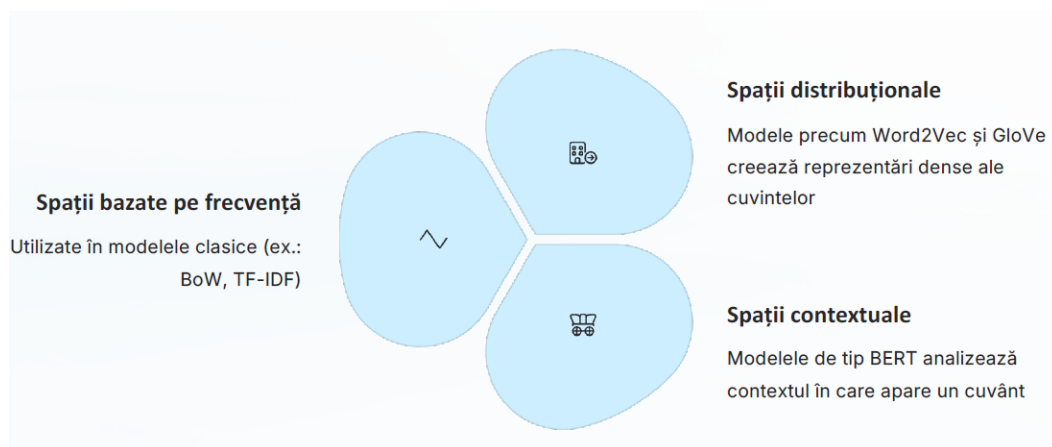


Fig. 8.8. *Principalele tipuri de spații de reprezentare a cuvintelor în NLP*

8.3. Metode de clasificare a textului

După transformarea textului într-o reprezentare numerică, aceasta poate fi utilizată într-un model de clasificare. Alegerea algoritmului depinde de dimensiunea datelor, natura reprezentării și complexitatea relațiilor care trebuie învățate. Pentru clasificarea de text pot fi utilizate atât regresia logistică, cât și SVM sau RNC.

Regresia logistică este un model simplu și eficient computațional pentru clasificarea binară. În NLP, poate fi combinată cu reprezentări precum TF-IDF și utilizată, de exemplu, pentru analiza sentimentelor sau filtrarea spamului. Un avantaj important este faptul că modelul furnizează probabilități, iar coeficienții săi pot fi interpretați relativ ușor. Limitarea principală este aceea că modelul descrie relații mai simple decât arhitecturile neuronale profunde și nu modelează direct contextul lung. Când reprezentarea textului

este bine aleasă și problema este relativ liniară, regresia logistică poate constitui o bază foarte eficientă.

SVM urmărește găsirea unui hiperplan care separă clasele într-un spațiu de caracteristici. Algoritmul este potrivit pentru reprezentări cu multe dimensiuni și funcționează bine în spații dispersate, precum cele produse de TF-IDF. Avantajele acestora sunt performanța bună pe date de dimensiuni mari și eficiența în spații cu multe dimensiuni. SVM poate fi o alegere solidă pentru seturi de date medii, însă timpul de antrenare poate crește pentru colecții mari. În plus, alegerea funcției de kernel devine importantă atunci când datele nu sunt separabile liniar.

RNC pot fi utilizate și pentru text prin aplicarea unor filtre locale asupra secvențelor de cuvinte sau embeddinguri. Ideea este similară celei întâlnite în procesarea imaginilor: filtrele detectează tipare locale. În cazul textului, aceste tipare pot corespunde unor n-gramme sau unor expresii relevante pentru clasificare. RNC sunt eficiente computațional și pot obține rezultate bune în special pentru clasificarea textelor scurte. După straturile convoluționale, pot fi utilizate operații de pooling și straturi complet conectate pentru decizia finală. Limitarea este că dependențele foarte lungi și relațiile globale dintre tokeni sunt mai greu de modelat direct [24].

8.4. Arhitecturi Transformers și mecanismul de auto-atenție

Modelele Transformer au schimbat paradigma procesării limbajului natural prin introducerea mecanismului de autoatenție. Spre deosebire de abordările care procesează strict secvențial informația, autoatenția permite unui token să ia în considerare direct celelalte tokenuri din aceeași secvență. Printre cele mai reprezentative modele bazate pe arhitectura Transformer se numără BERT, GPT și T5 (Figura 8.9).

Arhitectura Transformer pornește de la tokenizare și transformarea tokenilor în embeddinguri. Pentru ca modelul să păstreze informația referitoare la ordinea tokenilor, reprezentarea este completată cu informație pozițională. Reprezentările trec apoi prin blocuri repetate de atenție și feed-forward. Arhitectura Transformer utilizează, de asemenea, operații de normalizare și conexiuni de tip skip, care contribuie la stabilitatea și eficiența procesului de antrenare.



Fig. 8.9. *Principalele paradigme de modele Transformer*

Autoatenția permite modelului să acorde ponderi diferite tokenilor din context. Pentru reprezentarea unui anumit token, modelul calculează relații față de celelalte tokenuri, normalizează aceste scoruri și utilizează ponderile pentru a combina informația contextuală. Avantajul este posibilitatea de a surprinde relații pe distanțe mari în secvență, fără a depinde exclusiv de o procesare strict secvențială.

Mecanismul poate fi exprimat, la nivel general, prin trei reprezentări: query, key și value. Un scor de compatibilitate este calculat între query și key, iar ponderile rezultate sunt utilizate pentru agregarea vectorilor value. În cazul *multi-head attention*, sunt utilizate mai multe astfel de mecanisme în paralel, ceea ce permite captarea unor tipuri diferite de relații.

$$Attention(Q, K, V) = softmax\left(\frac{Q * K^T}{\sqrt{d_k}}\right) * V \quad (39)$$

După blocul de atenție, fiecare poziție este trecută printr-o rețea feed-forward aplicată independent token cu token. Rolul acestor straturi este de a introduce neliniaritate și de a recombina caracteristicile extrase de mecanismul de atenție. Într-o arhitectură Transformer, atenția și feed-forward-ul sunt repetate pe mai multe niveluri, rezultând reprezentări din ce în ce mai contextualizate.

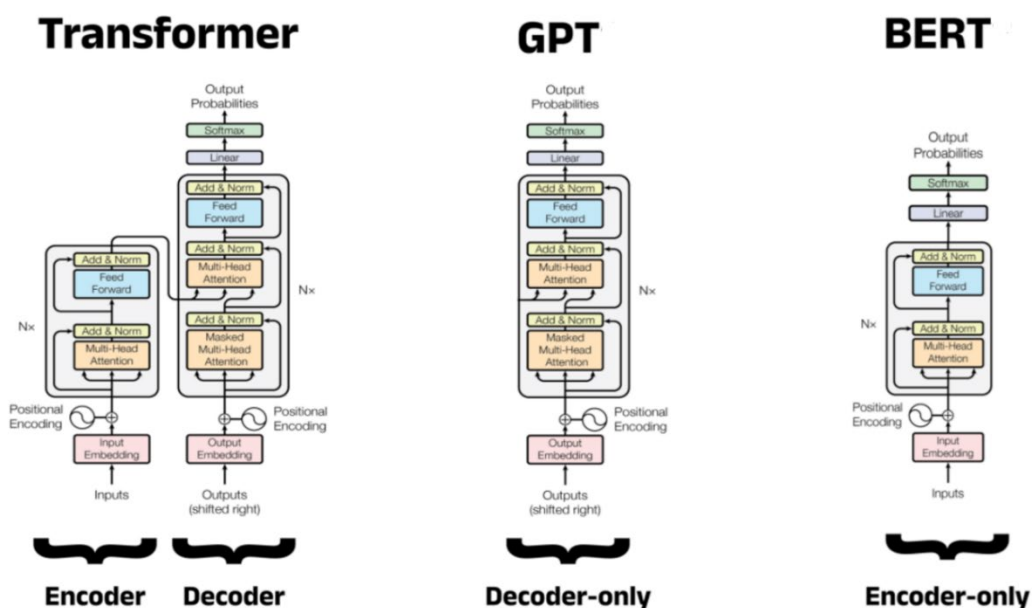


Fig. 8.10. Compararea arhitecturilor Transformer, GPT și BERT

Figura 8.10 prezintă o comparație structurală directă între arhitectura originală Transformer și cele două variații derivate principale ale sale, GPT și BERT.

- **Transformer:** prezintă o arhitectura completă de tip Encoder-Decoder propusă în lucrarea „Attention Is All You Need”. Aceasta combină un Encoder (stânga) care procesează secvența de intrare folosind Multi-Head Attention bidirecțional și un Decoder care generează secvența de ieșire folosind Masked Multi-Head Attention autoregresiv integrat cu mecanisme de atenție încrucișată (*cross-attention*).
- **GPT:** Prezintă o arhitectură Decoder-only (sau autoregresivă). Aceasta elimină blocul de Encoder și utilizează exclusiv blocuri de Decoder cu mecanism de atenție mascată (*Masked Multi-Head Attention*), fiind optimizată pentru generarea secvențială de text de la stânga la dreapta, prevăzând cuvântul următor.
- **BERT:** Ilustrează o arhitectură Encoder-only (sau bazată pe autoencoding). Elimină componenta de Decoder și utilizează un bloc de Encoder pur, permițând mecanismului de atenție (*Multi-Head Attention*) să proceseze bidirecțional întregul context al textului, fiind ideal pentru înțelegerea și clasificarea limbajului.

8.5. Aplicații NLP moderne

8.5.1. Bidirectional Encoder Representations from Transformers (BERT)

BERT (Bidirectional Encoder Representations from Transformers) este unul dintre modelele care au contribuit decisiv la trecerea de la reprezentări statice ale cuvintelor la reprezentări contextuale, în care semnificația unui cuvânt este determinată în funcție de cuvintele din jurul său. Modelul utilizează exclusiv componenta de encoder a arhitecturii Transformer și procesează simultan contextul aflat înainte și după fiecare token. Această caracteristică bidirecțională permite modelului să construiască reprezentări mai bogate decât abordările în care informația este analizată predominant într-o singură direcție.

Un exemplu simplu poate fi oferit de cuvântul „bancă”. În propoziția „Am depus banii la bancă”, termenul se referă la o instituție financiară, în timp ce în propoziția „M-am așezat pe bancă în parc”, sensul este diferit. Un model care asociază un singur vector fix cuvântului „bancă” nu poate reflecta în mod direct această diferență. BERT produce însă o reprezentare care depinde de contextul în care apare tokenul și, prin urmare, poate atribui reprezentări diferite aceluiasi cuvânt în propoziții diferite.

La baza funcționării BERT se află mecanismul de *self-attention*, prin care fiecare token poate lua în considerare celelalte tokenuri din secvență. Pentru un anumit cuvânt, modelul calculează cât de relevante sunt celelalte cuvinte pentru interpretarea sa și construiește o nouă reprezentare pe baza acestor relații. Procesul se repetă în mai multe capete de atenție (*multi-head attention*), fiecare cap putând învăța relații diferite. Unele capete pot surprinde relații sintactice, altele relații semantice sau de coreferență. Prin combinarea acestor informații se obține o reprezentare contextualizată a întregii secvențe.

Un element fundamental al pre-antrenării BERT este *Masked Language Modeling* (MLM). În această etapă, o parte dintre tokenii secvenței sunt ascunși, iar modelul trebuie să îi reconstruiască pe baza contextului disponibil. De exemplu, pentru propoziția „Modelul analizează datele textuale”, unul dintre tokeni poate fi ascuns, iar BERT trebuie să estimeze ce token este potrivit în poziția respectivă. Pentru a realiza această sarcină, modelul trebuie să utilizeze simultan informația din stânga și din dreapta tokenului ascuns. Această strategie obligă modelul să învețe relații contextuale și semantice complexe.

Pre-antrenarea este realizată pe corpusuri mari de texte, astfel încât modelul să învețe proprietăți generale ale limbajului înainte de a fi utilizat într-o aplicație concretă. Acest lucru reprezintă un avantaj important: modelul nu trebuie construit de la zero pentru fiecare problemă. După finalizarea pre-antrenării, BERT poate fi adaptat prin fine-tuning pentru o anumită sarcină și pentru un anumit set de date.

Într-o problemă de clasificare a textului, de exemplu, reprezentarea obținută pentru secvență poate fi transmisă unui strat final care determină categoria textului. Pentru analiza sentimentelor, BERT poate fi antrenat astfel încât propozițiile să fie încadrate în categorii precum pozitiv, negativ sau neutru. În cazul *Named Entity Recognition* (NER), modelul poate identifica tokenii care corespund unor persoane, organizații, localități sau alte tipuri de entități. BERT poate fi utilizat și pentru a oferi răspuns la întrebări, unde modelul identifică în textul sursă informația relevantă pentru o întrebare dată.

Un aspect important este faptul că BERT este orientat în primul rând către înțelegerea limbajului, nu către generarea liberă de text. Deoarece modelul utilizează contextul din ambele direcții, reprezentările sale sunt foarte utile atunci când este necesară interpretarea unei secvențe complete. Acest lucru îl face potrivit pentru clasificare, extragerea informațiilor și analiza semantică.

În același timp, utilizarea BERT presupune anumite costuri. Pre-antrenarea unui model de dimensiuni mari necesită resurse computaționale semnificative, iar procesarea secvențelor lungi este costisitoare. În plus, pentru limbi sau domenii specializate, performanța poate depinde de compatibilitatea dintre datele utilizate la pre-antrenare și domeniul aplicației. În practică, modele derivate din BERT pot fi adaptate prin **pre-**antrenare suplimentară pe corpusuri specializate sau prin fine-tuning.

Prin urmare, BERT reprezintă un punct de referință important în evoluția NLP deoarece a demonstrat eficiența utilizării reprezentărilor contextuale bidirecționale. În loc să trateze un cuvânt ca pe o entitate cu o singură reprezentare fixă, modelul construiește o reprezentare dependentă de context, ceea ce permite o interpretare mai precisă a limbajului natural.

8.6.2. Generative Pre-trained Transformer (GPT)

GPT (Generative Pre-trained Transformer) reprezintă o familie de modele bazate pe arhitectura Transformer, orientate în principal către generarea autoregresivă a textului. Spre deosebire de BERT, care utilizează partea de encoder, GPT este construit pe baza decoderului Transformer și utilizează o formă de autoatenție mascată. Astfel, pentru generarea unui token, modelul poate utiliza numai tokenii aflați anterior în secvență și nu are acces la informația care urmează să fie generată [18].

Caracterul autoregresiv al modelului poate fi explicat printr-un proces simplu. Dacă secvența de intrare este „Învățarea automată este”, modelul estimează distribuția probabilistică pentru următorul token și poate genera, de exemplu, „importantă”. Secvența devine apoi „Învățarea automată este importantă”, iar modelul estimează următorul token pe baza întregului context disponibil până în acel moment. Procedura continuă până când este atins un token de final sau o altă condiție de oprire.

Din punct de vedere matematic, modelul urmărește estimarea probabilității secvenței prin descompunerea autoregresivă:

$$P(x_1, x_2, \dots, x_n) = \prod_{t=1}^n P(x_t | x_1, x_2, \dots, x_{t-1}) \quad (40)$$

Această formulare evidențiază faptul că fiecare token este modelat condiționat de tokenii generați anterior. Prin antrenare pe corpusuri foarte mari, modelul învață regularități gramaticale, relații semantice, structuri textuale și numeroase tipare întâlnite în limbaj.

Un element esențial este *masked self-attention*. În timpul antrenării, accesul la tokenii viitori este blocat. Astfel, atunci când modelul învață să prezică tokenul de la poziția t , acesta poate folosi numai informațiile din pozițiile anterioare. Această restricție este necesară deoarece obiectivul modelului este tocmai învățarea procesului de continuare a unei secvențe.

GPT utilizează aceeași idee fundamentală a Transformerului descrisă anterior: tokenii sunt transformați în embeddinguri, se adaugă informația pozițională, iar reprezentările sunt procesate prin blocuri repetate de autoatenție și straturi feed-

forward. Conexiunile de tip skip și operațiile de normalizare contribuie la stabilitatea procesului de antrenare și permit construirea unor modele de dimensiuni foarte mari.

Una dintre caracteristicile definitorii ale GPT este capacitatea de a utiliza aceeași arhitectură pentru numeroase sarcini de generare de text. Modelul poate completa propoziții, redacta texte, rezuma informații, transforma un text într-un alt format și participa la dialoguri conversaționale. Într-o aplicație conversațională, mesajele precedente pot fi incluse în context, iar modelul generează răspunsul urmând aceeași regulă autoregresivă de predicție a tokenului următor.

Calitatea textului generat depinde nu numai de parametrii modelului, ci și de modul în care sunt selectați tokenii la fiecare etapă. Una dintre cele mai simple strategii este alegerea tokenului cu probabilitatea cea mai mare. Alte metode, precum temperature, top-k și top-p, modifică distribuția utilizată în procesul de generare și permit controlul gradului de diversitate al rezultatelor. O temperatură mai ridicată poate conduce la rezultate mai variate, în timp ce o temperatură redusă favorizează rezultate mai deterministe.

Un avantaj important al modelelor GPT este capacitatea de pre-antrenare pe corpusuri foarte mari, urmată de adaptarea la sarcini concrete. În loc să se construiască un model separat pentru fiecare aplicație, același model poate fi utilizat pentru mai multe probleme. Adaptarea poate fi realizată prin fine-tuning sau, în multe situații, prin formularea adecvată a cerinței într-un prompt.

În acest context apar abordări precum zero-shot și few-shot learning. În regim zero-shot, modelului îi este prezentată direct cerința, fără exemple suplimentare. În regim few-shot, în prompt sunt incluse câteva exemple de intrare și ieșire, iar modelul identifică tiparul și îl aplică noilor date. Aceste mecanisme permit adaptarea rapidă la sarcini pentru care nu a fost creat un model separat.

Modelele GPT prezintă însă și limitări. Fiind sisteme generative, ele pot produce răspunsuri care sunt formulate fluent, dar care conțin informații incorecte. În plus, modelele de dimensiuni mari presupun costuri ridicate de antrenare și inferență. Calitatea rezultatelor poate fi influențată și de datele pe care modelul a fost antrenat, inclusiv de eventualele erori și dezechilibre existente în acestea.

În ansamblu, GPT reprezintă paradigma decoder-only și autoregresivă a familiei Transformer. Dacă BERT este orientat în special către construirea unor reprezentări

contextuale pentru înțelegerea limbajului, GPT este optimizat pentru generarea succesivă a textului. Această diferență arhitecturală explică de ce cele două familii de modele sunt utilizate în contexte diferite, chiar dacă au la bază același mecanism fundamental de autoatenție.

8.6.3. Text-to-Text Transfer Transformer (T5)

T5 (Text-to-Text Transfer Transformer) propune o abordare diferită față de separarea strictă dintre sarcinile de înțelegere și cele de generare. Principiul central este transformarea tuturor sarcinilor NLP într-o problemă de tip text-la-text (*text-to-text*). Cu alte cuvinte, indiferent dacă problema inițială este clasificare, traducere, rezumare sau răspuns la o întrebare, atât intrarea, cât și ieșirea sunt reprezentate sub forma unor secvențe de text.

Această idee oferă o abordare unificată. De exemplu, într-o problemă de clasificare a sentimentului, intrarea poate fi formulată sub forma „clasifică sentimentul: filmul a fost excelent”, iar ieșirea așteptată poate fi „pozitiv”. În cazul traducerii, intrarea poate fi „tradu în engleză: Învățarea automată este importantă”, iar modelul produce propoziția corespunzătoare în limba țintă. În cazul rezumării, intrarea conține textul original și o instrucțiune de tipul „rezumă următorul text”, iar ieșirea este rezumatul.

T5 utilizează arhitectura encoder-decoder Transformer. Encoderul prelucrează întreaga secvență de intrare și construiește o reprezentare contextuală, în timp ce decoderul generează ieșirea token cu token. Decoderul utilizează atât autoatenție asupra tokenilor generați anterior, cât și atenție asupra reprezentării produse de encoder. Această organizare este potrivită în special pentru sarcini în care există o transformare explicită între o secvență de intrare și o secvență de ieșire.

Un avantaj important al acestei arhitecturi este uniformizarea sarcinilor. Într-un sistem tradițional, clasificarea, traducerea și sumarizarea pot necesita modele și interfețe diferite. În paradigma T5, aceeași arhitectură poate fi utilizată pentru toate aceste probleme. Diferența este exprimată prin modul în care este formulată intrarea și prin secvența de ieșire urmărită.

Modelul este pre-antrenat pe corpusuri de dimensiuni mari, ceea ce îi permite să învețe regularități generale ale limbajului. După pre-antrenare, T5 poate fi adaptat prin fine-tuning pe sarcina specifică. Această combinație dintre pre-antrenare și adaptare

specializată reduce necesitatea de a construi modele complet independente pentru fiecare aplicație.

Un exemplu important de sarcină este traducerea automată. Encoderul procesează propoziția din limba sursă și construiește reprezentări contextuale pentru tokeni. Decoderul generează apoi propoziția în limba țintă, utilizând reprezentarea encoderului și tokenii deja generați. În mod similar, în cazul sumarizării, encoderul primește textul complet, iar decoderul produce succesiv propoziția sau propozițiile care formează rezumatul.

Abordarea *text-to-text* este deosebit de interesantă și pentru clasificare. În loc să existe un strat final specializat care să producă direct o etichetă numerică, modelul poate genera eticheta sub formă de text. Astfel, categoria poate fi, de exemplu, „sport”, „politică” sau „tehnologie”. Procesul de instruire și evaluare rămâne în cadrul aceleiași paradigme generative. Un alt avantaj este flexibilitatea formatului de ieșire. T5 nu este limitat la răspunsuri de o singură clasă, ci poate genera secvențe de lungimi diferite. Acest lucru permite tratarea în aceeași arhitectură a unor sarcini care, în modelele clasice, ar necesita tratamente foarte diferite.

În același timp, această flexibilitate presupune și costuri. Deoarece ieșirea este generată token cu token, inferența poate fi mai costisitoare decât în cazul unor modele encoder-only utilizate pentru clasificare. În plus, rezultatul depinde de modul în care sarcina este formulată sub forma unui text de intrare și de calitatea datelor utilizate pentru fine-tuning.

T5 ilustrează astfel o etapă importantă în evoluția arhitecturilor Transformer: trecerea de la modele construite special pentru anumite categorii de sarcini la modele unificate, capabile să trateze probleme NLP foarte diferite prin același mecanism de bază. Conceptul *text-to-text* oferă o modalitate elegantă de a utiliza o singură arhitectură pentru clasificare, traducere, sumarizare, răspuns la întrebări și alte forme de transformare a textului.

8.6.4. Aplicații ale NLP modern

Reprezentările semantice și modelele Transformer sunt utilizate într-o varietate de aplicații. În motoarele de căutare și sistemele de recomandare, vectorii semantici permit

trecerea de la potrivirea lexicală simplă la căutarea bazată pe intenție. Query-ul utilizatorului și documentele pot fi mapate în același spațiu vectorial, iar similaritatea dintre reprezentări poate fi utilizată pentru ordonarea rezultatelor.

Figura 8.11 prezintă principalele aplicații practice ale reprezentărilor avansate din NLP și ale modelelor de limbaj:

- **Analiza Sentimentelor:** determinarea tonului emoțional al unui text.
- **Sisteme Întrebare-Răspuns:** furnizarea răspunsurilor relevante pe baza contextului furnizat.
- **Motoare de Căutare:** îmbunătățirea relevanței rezultatelor prin înțelegerea intenției căutării.
- **Traducere Automată:** generarea de traduceri contextuale precise între diferite limbi.
- **Recunoașterea Entităților:** identificarea și clasificarea entităților denumite (ex. nume de persoane, organizații, locații).
-



Fig. 8.11. *Principalele aplicații practice ale NLP*

Figura 8.12 ilustrează fluxul secvențial al unui sistem modern de căutare și recomandare pe bază de embedding-uri semantice, structurat în patru etape principale: transformarea mai întâi a interogării utilizatorului într-un vector dens ce-i captează intenția conceptuală, calcularea matematică a similarității dintre acesta și vectorii conținuturilor deja indexate, ordonarea ulterioară a rezultatelor în funcție de scorul de relevanță semantică obținut și, în final, ajustarea dinamică a ierarhiei pe baza istoricului și a preferințelor individuale ale utilizatorului.

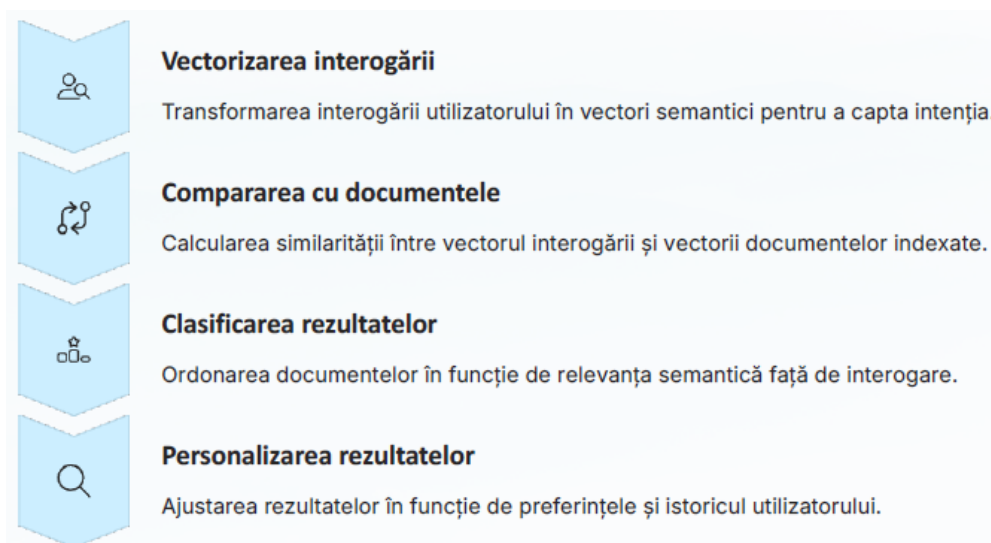


Fig. 8.12. *Utilizarea vectorilor semantici în căutare și recomandare*

În traducerea automată, arhitecturile encoder–decoder Transformer au îmbunătățit reprezentarea contextului și capacitatea de a produce secvențe în limba țintă (Figura 8.13). Encoderul codifică informația din limba sursă, iar decoderul generează textul corespunzător.



Fig. 8.13. *Utilizarea arhitecturilor Transformer în traducerea automată*

O altă direcție este clasificarea zero-shot și few-shot. În zero-shot, modelul poate trata clase care nu au fost prezentate explicit ca exemple etichetate în sarcina curentă, folosind descrierile și cunoștințele dobândite anterior. În abordarea few-shot, modelului îi sunt prezentate câteva exemple reprezentative, pe baza cărora acesta poate identifica tiparul și îl poate aplica unor exemple noi.

Aceste abordări sunt utile atunci când costul etichetării este mare sau când categoriile se schimbă frecvent. Zero-shot este potrivit pentru prototipare rapidă și clase dinamice, în timp ce few-shot poate crește performanța atunci când avem câteva exemple reprezentative.

Pentru clasificarea textului în timp real sunt necesare modele și pipeline-uri optimizate pentru latență redusă. Tehnici precum cuantizarea, eliminarea parametrilor neesențiali (*pruning*) și distilarea modelelor pot contribui la reducerea costului computațional al inferenței și la facilitarea rulării modelelor pe servere sau pe dispozitive de tip edge [12].

Figura 8.14 descrie arhitectura secvențială a unui sistem de clasificare a textului în timp real, structurată în patru etape optimizate pentru latență scăzută: preluarea continuă a datelor dintr-un flux de intrare (*streaming*), preprocesarea și vectorizarea rapidă a textului primit, executarea inferenței prin modele de clasificare optimizate computațional și, în final, actualizarea incrementală a sistemului pentru adaptarea continuă la noile modele de date.

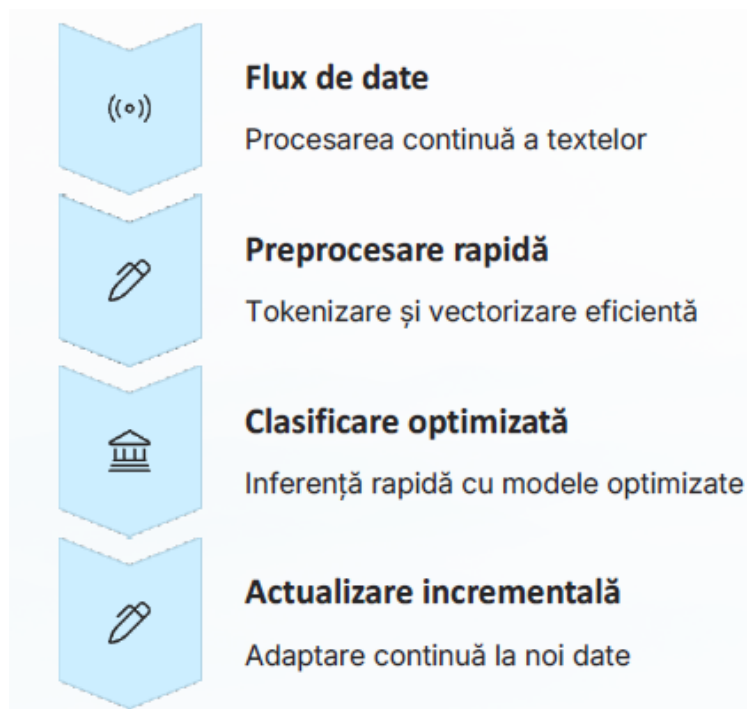


Fig. 8.14. Clasificarea textului în timp real

Alegerea unei arhitecturi pentru o aplicație reală este, astfel, un compromis între acuratețe, latență, cost de calcul, memorie și flexibilitate. Un model foarte complex poate obține rezultate bune, dar poate fi dificil de utilizat în regim de timp real. În schimb, un model mai compact poate fi preferabil atunci când răspunsul trebuie furnizat rapid [10].

Evoluția acestui domeniu poate fi rezumată ca trecerea de la reprezentări statice și modele specializate către reprezentări contextuale și arhitecturi unificate. În locul aplicării unei soluții distincte pentru fiecare problemă, modelele moderne sunt pre-antrenate pe volume masive de date și ulterior adaptate pentru sarcini specifice..

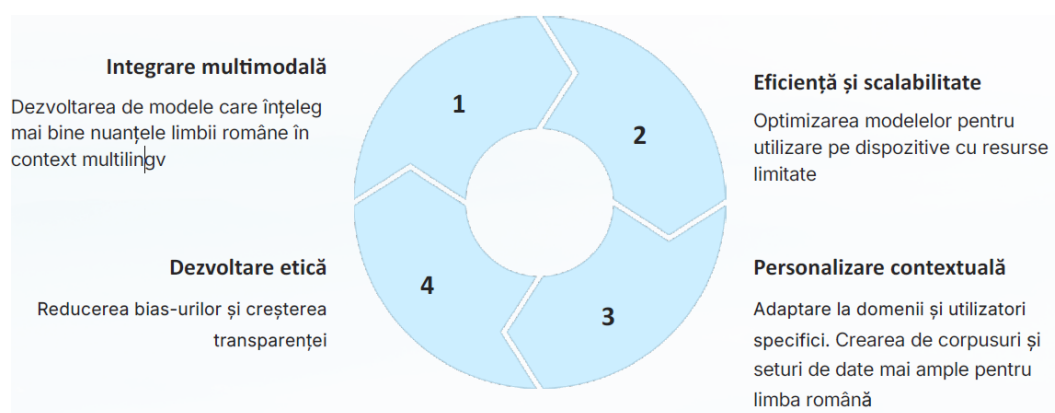


Fig. 8.15. *Viitorul NLP*

În același timp, performanța nu este singurul criteriu. Pe măsură ce modelele cresc, devin importante eficiența, interpretabilitatea, adaptarea la domenii și rularea pe resurse limitate. Pentru aplicații specializate este necesară alegerea unei reprezentări și a unei arhitecturi care să țină cont de limbă, domeniu, dimensiunea corpusului și cerințele de implementare.

Din perspectiva învățării automate, înțelegerea NLP presupune deci mai mult decât cunoașterea unui model. Este necesară parcurgerea întregului flux: preprocesarea și tokenizarea datelor, alegerea reprezentării, selectarea algoritmului, antrenarea și evaluarea, apoi optimizarea pentru condițiile reale de utilizare.

Bibliografie

1. Aggarwal, Charu C. 2023. *Neural Networks and Deep Learning: A Textbook*, 2nd Edition. Springer.
2. Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.
3. Chollet, François. 2021. *Deep Learning with Python*, 2nd Edition. Manning Publications.
4. Forsyth, D.A. & Jean Ponce, J. 2012. *Computer Vision: A Modern Approach*, 2nd Edition. Pearson.
5. Géron, Aurélien. 2022. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 3rd Edition. O'Reilly Media.
6. Goodfellow, I., Bengio, Y., & Courville A. 2016. *Deep Learning*. The MIT Press.
7. Gorunescu, Florin. 2011. *Data Mining: Concepts, Models and Techniques*. Springer. (Ediția originală în limba română a apărut la Editura Albastră, Cluj-Napoca, în 2006.)
8. Hastie, T., Tibshirani, R. & Friedman, J. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd Edition. Springer.
9. Hristea, Florentina. 2010. *Introducere în procesarea limbajului natural. Cu aplicații în Prolog*. București: Editura Universității din București.
10. Huyen, Chip. 2022. *Designing Machine Learning Systems: An Iterative Process for Production-Ready Applications*. O'Reilly Media.
11. Jurafsky, D. & Martin, J.H. 2026. *Speech and Language Processing*, 2nd Edition. Pearson.
12. Koul, A., Ganju, S. & Kasam, M. 2019. *Practical Deep Learning for Cloud, Mobile, and Edge: Real-World AI & Computer-Vision Projects Using Python, Keras & TensorFlow*. O'Reilly Media.
13. Leskovec, J., Rajaraman, A. & Ullman, J.D. 2020. *Mining of Massive Datasets*, 3rd Edition. Cambridge University Press.
14. Müller, A.C. & Guido, S. 2016. *Introduction to Machine Learning with Python: A Guide for Data Scientists*, 1st Edition. O'Reilly Media.
15. Murphy, Kevin P. 2021. *Probabilistic Machine Learning: An Introduction*. The MIT Press.

16. Murphy, Kevin P. 2023. *Probabilistic Machine Learning: Advanced Topics*. The MIT Press.
17. Popescu, D. & Flonta, M.L. 2009. *Teoria rețelelor neuronale artificiale*. Vol. 1. București: Editura Universității din București.
18. Raschka, S. & Mirjalili, V. 2019. *Python Machine Learning*, 3rd edition. Packt Publishing.
19. Raschka, S., Liu, Y. & Mirjalili, V. 2022. *Machine Learning with PyTorch and Scikit-Learn*. Packt Publishing.
20. Raschka, Sebastian. 2024. *Build a Large Language Model (From Scratch)*. Manning Publications.
21. Shalev-Shwartz, S. & Ben-David, S. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
22. Stancu, Stelian. 2020. *Machine Learning, Deep Learning, Deep Neural Network, utilizând mediul R. Teorie și aplicații*. București: Editura ASE.
23. Stancu, Stelian. 2022. *Data Mining în mediul R. Teorie și aplicații*. București: Editura ASE.
24. Tunstall, L., von Werra, L. & Wolf, T. 2022. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media.
25. Watt, J., Borhani, R. & Katsaggelos, A.K. 2020. *Machine Learning Refined: Foundations, Algorithms, and Applications*, 2nd Edition. Cambridge University Press.