



LUCHICI ANDREI

INTRODUCERE ÎN  
ÎNVĂȚAREA AUTOMATĂ  
Învățarea supravegheată și  
nesupravegheată în Python

București  
2023

*Pentru Mica, cea care a indurat cu stoicism multe dintre discuțiile anterioare realizării acestei lucrări.*

Luchici Andrei

Introducere în învățarea automată  
Învățarea supravegheată și  
nesupravegheată în Python

București  
2023

**Titlu:**

**Introducere în învățarea automata**

**Subtitlu:**

**Învățarea supravegheată și nesupravegheată în Python**

**ISBN:**

**978-973-0-37778-1**

## Cuprins

<b>1. INTRODUCERE .....</b>	<b>5</b>
BIBLIOGRAFIE.....	7
<b>2. INTRODUCERE IN ÎNVĂȚAREA AUTOMATĂ.....</b>	<b>8</b>
EXERCIIU .....	15
BIBLIOGRAFIE.....	15
<b>3. INTRODUCERE ÎN PYTHON .....</b>	<b>16</b>
<b>3.1. NOȚIUNI FUNDAMENTALE .....</b>	<b>16</b>
3.1.1. <i>Primul program în Python</i> .....	18
3.1.2. <i>Tipuri de date fundamentale</i> .....	19
3.1.4. <i>Introducere în funcții</i> .....	30
3.1.5. <i>Introducere în programarea orientată pe obiect folosind Python</i> .....	34
<b>3.2. PYTHON PENTRU ÎNVĂȚAREA AUTOMATĂ .....</b>	<b>40</b>
3.2.1. <i>Numpy</i> .....	41
3.2.2. <i>Scipy</i> .....	42
3.2.3. <i>Pandas</i> .....	43
3.2.4. <i>Scikit-learn</i> .....	44
3.2.5. <i>Matplotlib</i> .....	45
EXERCIIU .....	47
BIBLIOGRAFIE.....	48
<b>4.1. NOȚIUNI FUNDAMENTALE .....</b>	<b>48</b>
<b>4.2. BIAS SI VARIANȚĂ .....</b>	<b>63</b>
<b>4.3. K-NEAREST NEIGHBORS (KNN).....</b>	<b>68</b>
<b>4.4. REGRESIA LINIARA .....</b>	<b>71</b>
<b>4.5. REGRESIA LOGISTICĂ .....</b>	<b>76</b>
<b>4.6. ARBORI DE DECIZIE PENTRU CLASIFICARE SI REGRESIE .....</b>	<b>78</b>
<b>4.7. ANSAMBLURI.....</b>	<b>82</b>
EXERCIIU .....	87
BIBLIOGRAFIE.....	87
<b>5. ÎNVĂȚAREA NESUPRAVEGHEATA.....</b>	<b>89</b>
<b>5.1. NOȚIUNI FUNDAMENTALE .....</b>	<b>89</b>
<b>5.2. K-MEANS .....</b>	<b>92</b>
EXERCIIU .....	96
BIBLIOGRAFIE.....	97

# 1. Introducere

Învățarea automata este un domeniu interdisciplinar, aflat la intersecția dintre matematică, statistică și informatică. Învățarea automata ne ajută să dezvoltăm soluții computaționale pentru probleme din domenii foarte diverse folosind date despre problema pe care dorim să o rezolvăm, fără să cunoaștem alte detalii despre aceasta. Pentru a înțelege mai ușor ce este învățarea automată, vă propun să încercăm să rezolvăm o problemă.

Imaginați-vă că lucrați la o companie de marketing și aveți un client care v-a rugat să îl ajutați să promoveze un produs nou. Clientul vostru a reușit să lanseze pe piața produsul și a înregistrat câteva succese, reușind să vândă 100.000 de produse de la lansare. Mai mult, a fost prevăzător și a înregistrat toate detaliile despre cei care au achiziționat produsele, inclusiv date demografice și informații despre interacțiunea acestora cu site-ul de prezentare. În urma succesului înregistrat, clientul își dorește să se extindă, motiv pentru care realizează o companie de promovare cu mesaje diferite pentru fiecare din grupurile țintă de cumpărători.

Aici interveniți voi, fiind niște analiști foarte inventivi, îi cereți acces la datele colectate și imediat ce le aveți, începeți investigația. Prima dată vă faceți o idee despre conținutul datelor folosind o serie de metode statistice. După ce v-ați familiarizat cu datele, începeți să le curățați, eliminând cu atenție toate intrările eronate sau incomplete. O dată ce ați reușit să aveți un set de date curat, începeți munca de detectiv. După îndelungi ore petrecute făcând grafice, analizând corelații între variabile sau încercând diferite modele, ajungeți la concluzia că setul de date e prea divers fără să descoperiți nimic special legat de structura acelor date. Cu alte cuvinte, analiza manuală pe care ați efectuat-o nu a rezultat în nicio grupare informativă.

Imaginați-vă acum ca înlocuiți aceasta munca cu un program căruia îi oferiți toate datele și o metodă prin care să găsească automat grupuri de cumpărători. De exemplu, îi spuneți să caute în exemplele oferite un număr de patru grupuri, unde cumpărătorii dintr-un grup au caracteristici similare iar cumpărătorii din grupuri diferite au caracteristici diferite. Zis și făcut; învățarea automată vă oferă aceasta posibilitate.

Exemplul anterior constituie doar o posibilă aplicație a învățării automate. El reprezintă o categorie de probleme unde datele colectate nu sunt etichetate. Există însă situații când dorim să realizăm un algoritm care să poată identifica tipul unui potențial cumpărător, de exemplu, dacă acesta va finaliza sau nu comanda. Similar cu problema anterioară, și în aceasta situație, punctul de plecare îl reprezintă datele, cu mica diferență, și anume, de aceasta dată avem date despre toți vizitatorii magazinului nostru și pentru fiecare dintre ei avem o etichetă care indică dacă un vizitator a cumpărat sau nu ceva din magazin. Folosind învățarea automată vom putea crea un algoritm capabil să identifice care sunt cele mai importante trăsături ce diferențiază un client care va efectua o comandă de un client care doar „investighează” produsele companiei.

Aceste exemple sunt doar un preambul a ceea ce am putea realiza folosind învățarea automata. Ele au fost construite astfel încât să vă ajute să vă gândiți la ce ați putea realiza dacă stăpâniți

aceste unelte. Cu alte cuvinte, învățarea automată ne oferă o serie de unelte cu ajutorul cărora putem construi algoritmi (a se înțelege programe) capabili să efectueze o sarcină complexă fără a programa reguli specifice pentru a ajunge la rezultatul dorit.

Programarea „clasică” necesită o cunoaștere a rețetei prin care putem rezolva o anumită problemă. Acest lucru ne limitează posibilitățile atunci când vine vorba de soluțiile tehnice pe care le putem crea deoarece există foarte multe situații în care regulile sau pașii prin care rezolvăm o anumită problemă sunt vagi, incerti, sau chiar necunoscuți. Dacă nu sunteți convinși, imaginați-vă următoarea situație: cineva vă arată o poză cu o pisică; care sunt regulile pe care le urmați astfel încât să identificați obiectul din acea poză (pisica) și mai mult, să recunoașteți că acel obiect este într-adevăr o pisica? Această problemă, care în aparență pare trivială, este extraordinar de dificilă de implementat folosind „normele” programării „clasice”. În schimb, problema devine mult mai ușoară atunci când programăm o mașină să identifice diferența dintre o pisică și alt animal sau obiect din acea poză în baza unor exemple etichetate. Aici intervine învățarea automată. Acest domeniu ne oferă uneltele necesare pentru a putea instrui un calculator, sau un „gânditor” binar, despre cum poate să diferențieze între o pisica și alte componente ale imaginii. Nu vă imaginați acum că aceste unelte chiar ajută mașina să înțeleagă ce este în aceste poze. Nici pe departe; tot ce facem este să ajutăm un computer să diferențieze între conținutul unei imagini cu pisici și o imagine ce nu conține o pisică.

Poate va întrebați acum dacă acest sistem digital este capabil doar să identifice niște tipare, reprezintă el cu adevărat o mașinărie care „învăță”? Poate aceasta întrebare vă conduce la o altă, și anume, ce anume înseamnă a învăța? Există oare o definiție obiectivă pentru acest concept pe care să o putem transforma în metode matematice sau numerice?

Aceste întrebări stau la baza învățării automate. În prezent, nu avem un răspuns definitiv asupra acestei probleme. Însă, avem o colecție de idei și de metode prin care putem realiza sisteme computaționale capabile să „învețe” diferite „tipare”. În continuare veți regăsi câteva metode cu ajutorul cărora putem crea algoritmi capabili să identifice anumite tipare cu ajutorul unor exemple etichetate sau nu.

În concluzie, învățarea automată este un instrument pentru crearea de algoritmi care rezolvă sarcini folosind date. Este un subdomeniu al inteligenței artificiale (AI) care utilizează algoritmi pentru a identifica modele în date și pentru a face predicții sau a lua decizii pe baza acestor modele. În general, algoritmii de învățare automată folosesc tehnici statistice pentru a găsi modele în date. Învățarea automată poate fi utilizată pentru o varietate de sarcini, cum ar fi prezicerea prețurilor acțiunilor, detectarea fraudelor sau segmentarea clienților.

În esență, învățarea automată se referă la programarea computerelor astfel încât acestea să învețe din experiențele lor. Ideea este că, prin expunerea computerului la un set suficient de mare de date, acesta poate învăța cum să recunoască tipare și să ia decizii sau să facă predicții despre date noi. Acest lucru se realizează prin algoritmi care sunt optimizați pentru o anumită sarcină sau problemă. Algoritmii de învățare automată pot fi supravegheați sau nesupravegheați, în funcție de tipul de sarcină sau de problema rezolvată. Algoritmii de

învățare supravegheată sunt utilizați atunci când datele au etichete sau categorii, în timp ce algoritmi de învățare nesupravegheată sunt utilizați atunci când datele nu au etichete.

Dacă sunteți curioși, parcurgeți această lucrare pentru a afla mai multe detalii despre cum putem crea asemenea programe folosind exemple etichetate sau exemple fără etichete. Lucrarea de față a fost realizată ca suport de curs pentru un curs introductiv în învățarea automată pentru studenții din anul doi sau trei ce urmează un profil tehnic.

Lucrarea este împărțită în cinci capitole. Al doilea capitol vă oferă o introducere în învățarea automată. Acesta este urmat de un capitol dedicat limbajului de programare Python. Dacă sunteți deja familiarizați cu acest limbaj puteți să continuați cu capitolul următor. În capitolul 4 vom discuta despre învățarea supravegheată și vom vedea câteva dintre metodele folosite pentru a realiza soluții de învățare pornind de la un set de date etichetate. În final, capitolul 5 ne introduce în lumea învățării nesupravegheate și ne va arata un algoritm cu ajutorul căruia putem descoperi tipare în date neetichetate.

## Bibliografie

- Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC. ISBN 0387310738.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer Science+Business Media. ISBN 9781461471370.
- Jang, J.-S. R., Sun, C. T., & Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*. Upper Saddle River: Prentice Hall, Inc. ISBN 9780132610667.
- Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press. ISBN 9780262018029.
- Negnevitsky, M. (2005). *Artificial Intelligence - A guide to intelligent systems*. Pearson Education Limited. ISBN 0321204662.
- Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press. ISBN 9781107057135.
- Sivanandam, S. N., & Deepa, S. (2011). *Principles of Soft Computing*. New Delhi: Wiley India Pvt Ltd. ISBN 9788126527410.



## 2. Introducere în învățarea automată

Sistemele expert<sup>1</sup> și soluțiile cu rețete pre-programate și-au atins apogeul destul de repede. Cu toate că am reușit să realizăm multe aplicații de succes folosindu-ne de aceste abordări, astăzi observăm din ce în ce mai mult nevoia unor soluții capabile să îndeplinească anumite sarcini fără a fi nevoie de programarea explicită a modului în care acestea sunt îndeplinite. În plus, sistemele expert sunt foarte greu de adaptat pentru a face față situațiilor neprevăzute în corpul de cunoștințe sau în regulile implementate. Pe măsură ce dezvoltăm aceste sisteme, ne dăm seama că există numeroase probleme și situații a căror rezolvare nu o putem transpune într-un set de reguli. Acest lucru se datorează unui fapt simplu, și anume, nu știm care sunt acele reguli.

Gândiți-vă la o problemă simplă. Aruncați o privire la imaginile din figura 1 și numărați câte dintre imaginile din această figură conțin o pisică. Aceasta problemă nu reprezintă nici o dificultate pentru noi. În câteva secunde, poate chiar milisecunde, putem nu doar să numărăm câte imagini conțin pisici (sunt doar două imagini cu pisici în cazul în care nu erați siguri) dar am și identificat ce conțin și celelalte imagini. Noi putem rezolva aceste probleme deoarece în timpul dezvoltării noastre am fost învățați sau am învățat singuri cum să le rezolvăm. Prin analiza unor exemple sau prin încercări repetate, noi, oamenii, am dezvoltat diferiți algoritmi cu ajutorul cărora putem îndeplini sarcini diverse, chiar și sarcini pe care nu le-am întâlnit niciodată în trecut.

Însă, această sarcină, și ca ea mai sunt multe altele, este extrem de dificilă<sup>2</sup> de realizat doar prin intermediul un sistem expert sau folosind o abordare algoritmică clasică.

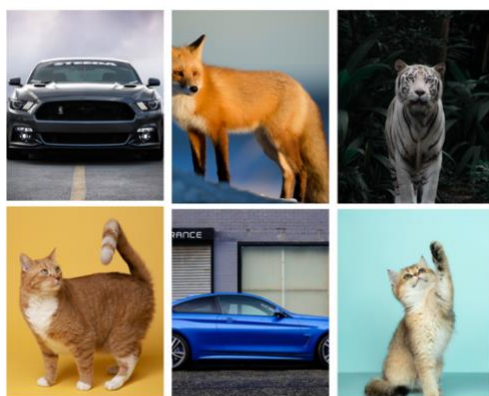


Figura 1. Șase imagini ce conțin diferite obiecte și animale. Creierul nostru nu are nici o problemă în a interpreta și înțelege ce obiect sau animal se află în fiecare imagine, chiar dacă acestea sunt reprezentate în diferite unghiuri sau dimensiuni. În schimb, este extrem de dificil să implementăm un algoritm bazat pe reguli predefinite care să aibă aceeași abilitate de a înțelege ce conține fiecare imagine și de a generaliza (a fi capabil să recunoască diferite obiecte sau animale).

<sup>1</sup> [https://sim.tuiasi.ro/wp-content/uploads/Vizureanu-Expert\\_Systems.pdf](https://sim.tuiasi.ro/wp-content/uploads/Vizureanu-Expert_Systems.pdf)

<sup>2</sup> În contextual actual. În viitor poate vom descoperi care sunt regulile prin care să rezolvăm asemenea probleme și atunci vom putea avea sisteme expert capabile să recunoască orice obiect.

Gândiți-vă la o altă situație. Imaginați-vă că urmărim un proces, de exemplu resortul reprezentat în figura 2. Pe măsură ce observăm întinderea și contractarea acestui resort notăm pe o foaie de hârtie care este alungirea resortului la intervale fixe de timp. Aceste observații constituie un set de date<sup>3</sup> despre modul de desfășurare a acestui proces. Plecând de la aceste observații dorim să construim o soluție computațională care să poată descrie traiectoria resortului indiferent de proprietățile fizice ale acestuia, de datele de intrare ale problemei, sau de mediul înconjurător în care se regăsește resortul. Din punctul meu de vedere ar fi extrem de dificil să întreprindem așa ceva. Imaginați-vă doar câte tipuri de resorturi există, cât de variate sunt intrările și câte condiții de mediu putem avea.

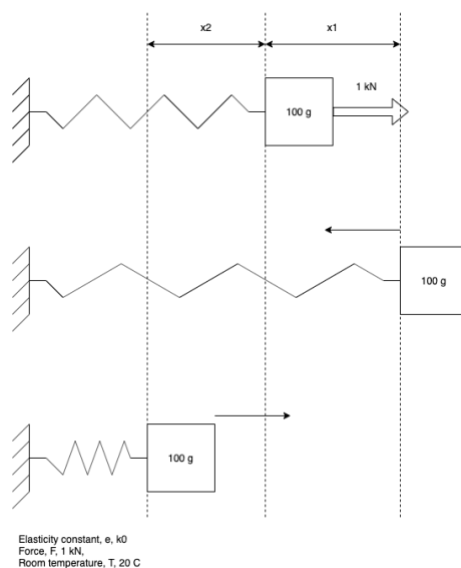


Figura 2. Un exemplu al unei mase și a unui arc sub acțiunea unei forțe,  $F$ , de 1 kN.

Diagrama reprezintă un exemplu de proces pe care am dori să îl putem înțelege sau descrie folosind un sistem expert. În special, suntem interesați să putem obține distanța parcursă de masa de 100g la orice moment de timp după ce acțiunea forței  $F$  s-a oprit. După cum se poate observa din diagramă, descrierea modului în care valorile observate ale lui  $x$  ( $x_1$  și, respectiv,  $x_2$ ) vor necesita multe experimente și va fi incredibil de dificilă.

Dacă exemplele de mai sus nu vă conving că soluțiile algoritmice clasice nu reprezintă o alternativă viabilă pentru orice situație, vă propun să analizăm încă un exemplu. Imaginați-vă că repetăm experimentul cu resortul de mai sus doar că de data aceasta vom măsura poziția masei sau lungimea acestuia cu un instrument de măsură inexact. Acest instrument de măsură are o eroare variabilă între  $-1\text{ cm}$  și  $1\text{ cm}$ . Nu știm cu exactitate ce eroare de măsurare vom avea la fiecare măsurătoare. În schimb, știm că valoarea erorii este distribuită conform distribuției din figura 3. Folosind acest instrument este evident că vom avea niște măsurători incerte. Pe baza acestor măsurători, trebuie să găsim un set de reguli pentru a explica sau pentru a prezice fiecare observație. În acest moment, sarcina noastră a devenit extrem de complicată într-un timp extrem de scurt.

<sup>3</sup> Pentru moment gândiți-vă la date ca la niște valori ale unor caracteristici pe care le înregistrați undeva. De exemplu, dacă vorbim despre o persoană, putem să considerăm numele, adresa, înălțimea, locul de munca, profesia, și alte detalii despre această persoană ca fiind niște date pe care le putem folosi pentru a o descrie.

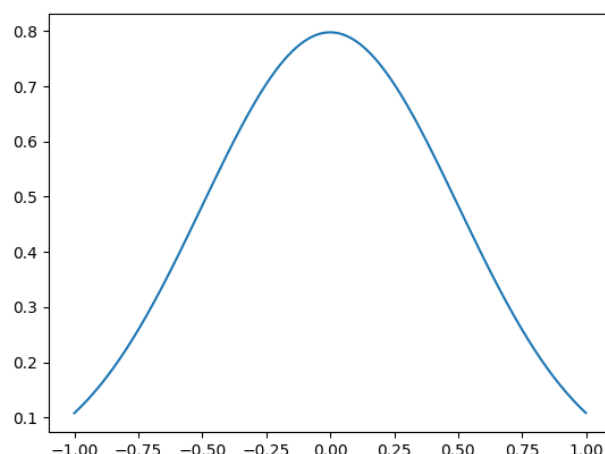


Figura 3. Exemplu al distribuției erorilor de măsurare. Graficul reprezintă probabilitatea ca eroarea să aibă o anumită valoare între  $-1\text{ cm}$  și  $1\text{ cm}$ .

Din aceste motive și multe altele, cercetătorii și inginerii au început să caute alternative pentru sistemele expert și pentru soluțiile algoritmice clasice. În căutările lor au descoperit o idee salvatoare. Ce ar fi dacă în loc să încercăm să găsim noi, oamenii, regulile și ulterior să le programăm, am construi un program capabil să găsească regulile cu ajutorul unor exemple?

Această idee a fost inspirată și de munca depusă în domeniul inteligenței artificiale, unde ne dorim să realizăm un sistem capabil să manifeste comportamente inteligente în orice situație. Și acolo, cercetătorii s-au „împotmolit” cu abordările clasice. Din fericire, au realizat rapid că o posibilă soluție pentru problemele cu care se confruntau ar putea fi învățarea.

În general organismele vii nu se nasc neapărat capabile de toate comportamentele inteligente pe care le manifestă de-a lungul vieții. În cele mai multe cazuri acestea au doar un set limitat de comportamente pe care le folosesc ulterior, pe durata vieții, să achiziționeze alte comportamente inteligente, inclusiv alte moduri de gândire sau de rezolvare a problemelor cu care confruntă. Natura, ajutată de o perioadă destul de îndelungată de evoluție, a reușit să dezvolte diferite metode prin care un organism să poată acumula cunoștințe și/sau comportamente noi, prin care să învețe.

Acest lucru este evident în cazul nostru, al oamenilor. Când ne naștem nu știm mai nimic. Însă, pe măsură ce ne maturizăm și înaintăm în vârstă și prin expunerea la diferite situații, ajungem să dobândim foarte multe cunoștințe și să învățăm să întreprindem tot felul de acțiuni și activități complexe. Mai mult, tot prin intermediul învățării și al altor mecanisme conexe, suntem capabili nu doar să rezolvăm sarcini complexe, ci și să ne folosim de toate datele acumulate anterior pentru a genera idei noi sau pentru a explica modul în care funcționează lumea înconjurătoare; uneori chiar înfricoșător de bine sau de detaliat (vezi de exemplu mecanica cuantică și toate tehnologiile contemporane, inclusiv dispozitivul de pe care citiți aceste rânduri).

Imaginați-vă cum ar arata viața voastră dacă ați avea o mașină capabilă să vă rezolve orice problemă sau cel puțin să vă ofere un rezumat inteligibil al datelor problemei cu care vă confrunțați astfel încât să vă puteți concentra doar pe lucrurile esențiale în rezolvarea acestora. Nu ar fi mult mai ușor să navigăm prin lumea înconjurătoare având la dispoziție un asemenea sistem capabil să ne recomande ce vrem să citim, ce filme să vizionăm, ce decizii să luăm în situații critice? Cum ar fi să avem un asistent de studiu sau de laborator capabil să îngereze toată cunoașterea omenirii și pe care să îl putem interoga punctual despre subiectele care ne interesează? Din punctul meu de vedere, cred că am putea realiza mult mai multe lucruri asistați de asemenea sisteme decât o facem astăzi și nu doar pentru că ar prelua multe din sarcinile noastre plictisitoare care ne consumă mult timp, ci și din prisma faptului că împreună am putea găsi rezolvări la multe din problemele presante ale omenirii. Dar, există și reversul medaliei. Această mașinărie ar putea să ne facă să ne simțim atât de confortabili încât să nu dorim să mai depunem nici un efort. Nu cred că o asemenea mașină este un lucru de dorit dacă ar conduce la a doua variantă.

Din fericire însă, mai avem până când vom putea realiza o asemenea mașinărie, prin urmare, ne putem gândi fără teamă (dar fiind conștienți de potențialele consecințe) la ce anume am dori să realizeze această mașinărie minune. Cum ar arata înțelegerea de care ar trebui să dea dovadă o asemenea mașină și cum ar putea aceasta să învețe?

În cazul oamenilor, înțelegerea unei situații sau a unui fenomen se manifesta în mai multe moduri, cea mai comună fiind o reprezentare idealizată a principalelor evenimente sau componente ale fenomenului și a relațiilor dintre aceste componente în timp și spațiu. Dacă suntem norocoși (sau dacă fenomenul este suficient de „simplu”) reușim chiar să transpunem aceasta reprezentare într-o formă simbolică, de multe ori, o formă matematică.

Nu trebuie să mergem prea departe pentru a găsi câteva exemple. Gândiți-vă la exemplul din figura 2, al resortului ce își schimbă lungimea sub acțiunea unei forțe. De-a lungul timpului oamenii de știință, fizicieni și matematicieni, au analizat componentele acestui sistem (și altele similare) reușind, în urma multor observații și a unui proces de gândire îndelungat, presărat cu multe erori și drumuri înfundate, să formuleze un set de legi prin intermediul cărora să putem descrie comportamentul aceluia resort în diferite grade de detaliu.

Alteori nu avem însă acest lux și trebuie să ne mulțumim cu o poveste, o descriere lingvistică sau pictorială. Nici în această situație nu trebuie să ne gândim prea profund să găsim câteva exemple. Gândiți-vă la majoritatea descoperirilor de la începutul biologiei (chiar și o parte din descoperirile contemporane), teoriile din psihologie, sociologie și alte științe care se ocupă cu studiul materiei animate (vie). A nu se înțelege că nu avem și aici reprezentări matematice. Departe de adevăr acest lucru. Însă, prin comparație cu materia neanimată, nu avem un set de legi precise prin care să putem explica sau de la care să putem formula deducții logice ce explică multitudinea observațiilor făcute de cercetători de-a lungul timpului.

Într-adevăr, aceasta situație este parțial cauzată de complexitatea fenomenelor și proceselor analizate, care au un număr foarte mare de variabile pe care nu le putem măsura direct, fiecare

cu efecte dependente de scară sau care au numeroase proprietăți emergente. Mai mult, când vorbim despre materia animată, suntem de cele mai multe ori restricționați când facem măsurători precise deoarece instrumentele existente nu ne permit acest lucru (de exemplu, nu putem să urmărim ce se întâmplă cu fiecare proteină din interiorul fiecărui neuron în timp ce un organism îndeplinește o sarcină). Imaginați-vă ce ar însemna să putem studia dezvoltarea creierului de la formarea acestuia până la maturitate și chiar după aceea. Am avea nevoie de instrumente capabile să măsoare simultan toate procesele biofizice și celulare ce au loc. Dacă acest lucru nu vi se pare greu, imaginați-vă că ar trebui să efectuăm aceste măsurători în interiorul unui organism viu care trăiește și se dezvoltă în mediul său, înconjurat de alte organisme similare.

Ați putea argumenta acum că aceleași situații se regăsesc și în cazul studierii materiei neanimate, nu mai departe de studierea originii universului sau a găurilor negre. Nu vă înșelați. Și acolo avem o problemă similară. Însă, regulile fizicii descoperite până acum ne oferă o explicație destul de rezonabilă pentru observațiile pe care le facem.

Dacă totul e atât de complicat, ce soluții ne rămân pentru a înțelege aceste procese și fenomene unde nu avem nici cea mai mică șansă de a măsura în mod direct ce se întâmplă? Din punctul meu de vedere, nu trebuie să ne sperie aceste situații, chiar aș recomanda să le îmbrățișăm deoarece ne oferă șansa să descoperim lucruri interesante despre lumea înconjurătoare. În timp, comunitatea științifică a dezvoltat metode prin care să investigăm fenomene unde nu avem acces direct la parametrii pe care dorim să îi măsurăm. Punctul de plecare în aceste situații îl constituie observarea unor fenomene, procese sau manifestări conexe. Cu alte cuvinte, în loc să ne uităm la toate componentele sistemului și să le urmărim în detaliu, observăm alte componente la care avem acces. Bineînțeles că aici facem un salt destul de mare deoarece presupunem că acele componente pe care le măsurăm au legătură directă cu fenomenul studiat.

De exemplu, pentru a investiga cum reacționează neuronii la anumiți stimuli, măsoară ce se întâmplă cu ionii de calciu înainte, în timpul și după stimularea acestora<sup>4</sup>. Aceste observații ne oferă o bază de la care plecăm; analizându-le și coroborând rezultatele cu alte rezultate sau măsurători formulăm și validăm ipoteze legate de modul de funcționare al creierului în diferite situații. În acest fel putem să începem să răspundem la diferite întrebări ce, cum sau de ce legate de dezvoltarea și funcționarea creierului.

De fiecare dată când reușim să confirmăm o ipoteză mai facem un pas spre a dezvălui misterele fenomenului investigat. Dar ce se întâmplă când nu reușim acest lucru? În caz contrar, reluăm procesul sau anumiți pași din proces. Dacă suntem suficient de perseverenți și avem la dispoziție resursele materiale și umane necesare, reușim până la urmă să găsim o explicație nouă despre cum funcționează procesul sau fenomenul respectiv.

Toate aceste activități conduc la o reprezentare idealizată a sistemului sau a fenomenului investigat. Această reprezentare poate avea mai multe scopuri. De exemplu, putem crea o

---

<sup>4</sup> <https://pubmed.ncbi.nlm.nih.gov/24442513/>

reprezentare cu ajutorul căreia să descriem sistemul și observațiile pe care le facem asupra sistemului sau o reprezentare care să ne ajute să facem predicții despre stările trecute sau viitoare ale aceluiași sistem. În știință aceste reprezentări poartă generic denumirea de modele. În funcție de metodele folosite pentru a le dezvolta, modelele pot fi matematice, mentale sau vizuale.

Modelele matematice sunt reprezentări ale sistemelor sau proceselor folosind concepte și ecuații matematice. Ele sunt folosite pentru a descrie și înțelege fenomene complexe, pentru a face predicții și pentru a testa ipoteze. Modelele matematice pot lua multe forme, inclusiv modele analitice, modele statistice și modele de calcul.

Modelele analitice implică utilizarea ecuațiilor matematice pentru a reprezenta relațiile dintre diferite variabile dintr-un sistem. Aceste modele pot fi folosite pentru a rezolva probleme și a face predicții, dar pot necesita un nivel ridicat de abstractizare și pot să nu reflecte întotdeauna cu acuratețe sistemele din lumea reală.

Modelele statistice implică utilizarea tehnicilor statistice pentru a analiza datele și a face predicții. Aceste modele sunt adesea utilizate în domenii precum economie și finanțe, unde sunt disponibile cantități mari de date și poate exista o incertitudine semnificativă cu privire la relațiile de bază dintre variabile.

Modelele computaționale implică utilizarea simulărilor pe computer pentru a reprezenta sisteme și procese complexe. Aceste modele pot fi folosite pentru a face predicții și a testa ipoteze și sunt adesea folosite în domenii precum meteorologie, inginerie și biologie.

În contextul învățării automate, un model este o reprezentare matematică a unui sistem sau proces care poate fi folosit pentru a face predicții sau decizii. În alte domenii, un model se poate referi la o reprezentare simplificată a unui sistem sau concept care este folosit pentru a înțelege sau explica anumite fenomene.

De exemplu, în învățarea automată, un model poate fi antrenat pe un set de date de date istorice, cum ar fi modelele de cumpărare ale clienților sau prețurile acțiunilor, și apoi utilizat pentru a prezice rezultatele viitoare sau pentru a lua decizii pe baza noilor date de intrare. În biologie, un organism model este o specie care este utilizată pe scară largă în cercetare ca reprezentant al unui grup mai mare de organisme. În inginerie, un model poate fi o reprezentare fizică sau matematică a unui sistem, cum ar fi un model la scară a unei clădiri sau o simulare pe computer a unei aeronave.

Modelele mentale sunt o reprezentare cognitivă a unui concept sau a unui sistem care ajută o persoană să înțeleagă și să prezică cum se va comporta. Modelele mentale sunt construite din experiențele, cunoștințele și observațiile unei persoane și sunt folosite pentru a simplifica și interpreta lumea din jurul nostru.

De exemplu, o persoană poate avea un model mental al modului în care funcționează o mașină, care îi permite să înțeleagă cum funcționează diferitele părți ale unei mașini și cum să o conducă. Modelele mentale sunt adesea folosite pentru a face predicții, cum ar fi prezicerea modului în care va răspunde o mașină atunci când este apăsată accelerația sau cum va funcționa un anumit mecanism.

Modelele mentale pot fi utile în înțelegerea și navigarea sistemelor complexe, dar pot fi și limitative dacă se bazează pe informații incomplete sau incorecte. Este important să actualizați și să revizuiți în mod continuu modelele mentale pe măsură ce noi informații devin disponibile.

Modelele vizuale sunt o reprezentare a unui concept sau a unui sistem care utilizează elemente vizuale precum imagini, diagrame și diagrame pentru a comunica informații. Acestea pot fi folosite pentru a ilustra idei complexe, procese sau relații într-un mod ușor de înțeles și de reținut prin comparație cu modelele matematice.

Modelele vizuale pot lua multe forme, inclusiv diagrame, hărți mentale, diagrame, infografice și diagrame. Ele pot fi folosite pentru a reprezenta o gamă largă de concepte, inclusiv concepte științifice, procese de afaceri, evenimente istorice și multe altele. Modelele vizuale sunt adesea folosite în educație, marketing și design pentru a comunica informațiile în mod eficient și a implica spectatorul.

În cazul nostru, ne vom ocupa cu precădere de modelele matematice și de cele folosite în învățarea automată. Chiar dacă scopul nostru este să învățăm cum să realizăm modele matematice aceasta nu înseamnă ca nu ne vom folosi și de celelalte tipuri de modele. Din experiența mea, un model matematic este digerat mai ușor dacă avem o înțelegere intuitivă a comportamentului sistemului, fenomenului sau procesului. Pentru a ajunge în acel punct avem nevoie de modele mentale și vizuale pe care ulterior le vom formaliza folosind ecuații și metode computaționale.

În restul acestei lucrări ne vom preocupa cu mai multe forme de învățare automată, fiecare specializată pentru o anumită sarcină. În general vom întâlni probleme de învățare unde avem exemple etichetate (învățare supravegheată) și exemple fără etichete (învățare nesupravegheată). Însă, acestea nu sunt singurele tipuri de sarcini unde învățarea automată poate fi folosită. Alte capitole importante din învățarea automată sunt învățarea prin întărire<sup>56</sup>, învățarea semi-supravegheată<sup>7</sup> sau învățarea activă<sup>8</sup>.

În concluzie, învățarea automată ne oferă o serie de metode cu ajutorul cărora putem realiza modele matematice și computaționale plecând de la o serie de exemple (observații). Este un domeniu de studiu relativ nou, care are potențialul de a revoluționa modul în care facem știință și rezolvăm probleme. Cu ajutorul tehnicilor de învățare automată, putem învăța din date, le

---

<sup>5</sup> <https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/>

<sup>6</sup> <https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292>

<sup>7</sup> <https://lilianweng.github.io/posts/2021-12-05-semi-supervised/>

<sup>8</sup> <https://www.datarobot.com/blog/active-learning-machine-learning/>

putem analiza și crea modele care pot fi utilizate pentru analiza predictivă și luarea deciziilor. Acest lucru poate fi extrem de util într-o gamă largă de aplicații, cum ar fi asistența medicală, finanțe și marketing.

În general, învățarea automată este o nouă ramură interesantă a științei, care are potențialul de a revoluționa modul în care abordăm rezolvarea problemelor și analiza datelor. Este important de reținut că astfel de tehnici necesită o analiză și o implementare atentă pentru a fi utilizate în mod eficient și responsabil. Pe măsură ce tehnologia continuă să avanseze, la fel vor continua și posibilitățile de învățare automată.

## Exerciții

1. Sumarizați ce înțelegeți prin învățarea automată.
2. Care sunt principalele diferențe dintre învățarea supervizată și cea nesupervizată?

## Bibliografie

- Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC. ISBN 0387310738.
- Deisenroth, M., Faisal, A., & Ong, C. (2020). *Mathematics for Machine Learning*. Cambridge University Press. ISBN 9781108455145.
- Jang, J.-S. R., Sun, C. T., & Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*. Upper Saddle River: Prentice Hall, Inc. ISBN 9780132610667.
- Negnevitsky, M. (2005). *Artificial Intelligence - A guide to intelligent systems*. Pearson Education Limited. ISBN 0321204662.



## 3. Introducere în Python

### 3.1. Noțiuni fundamentale

În acest capitol vom afla despre un limbaj de programare extrem de folosit pentru a dezvolta algoritmi care învață automat, și anume, Python. Acest limbaj de programare ne ajută să implementăm metodele și să realizăm unelte cu ajutorul cărora putem crea programe care rezolvă probleme folosindu-se de învățarea automată. Dacă sunteți deja familiari cu limbajul de programare și librăriile folosite pentru dezvoltarea aplicațiilor de calcul științific (NumPy, SciPy, Pandas, Matplotlib) puteți să săriți acest capitol fără a pierde continuitatea materialului.

Python este un limbaj high-level, interpretat, dinamic, generic (acest limbaj de programare poate fi folosit pentru a implementa orice tip de program sau aplicație). Python este un limbaj high-level deoarece nu manipulăm direct memoria calculatorului ci ne bazăm pe un interpretor să realizeze acest lucru pentru noi. Unul din avantajele acestui limbaj de programare este că ne permite să ne concentrăm mai mult pe programele pe care le dezvoltăm, fără a fi distrași de probleme legate de alocarea resurselor sau managementul memoriei disponibile pentru a rula programele noastre. Mai mult, un program scris în Python nu necesită compilarea codului înainte a fi rulat deoarece avem la dispoziție un interpretor care citește și traduce codul scris de noi în cod mașină în timpul execuției programului. Translația dintre codul scris de noi și limbajul pe care îl înțelege mașinăria pe care rulăm acest cod are loc în timp real, după ce programul a început să ruleze. Dacă acest lucru nu v-a intrigat, imaginați-vă ca în Python nu este nevoie să specificăm tipul unei variabile! Interpretorul își dă seama în timp real ce tip are o variabilă și îi alocă memoria necesară! Foarte tare, nu?

Python este un limbaj orientat pe obiect. În Python totul este un obiect, ceea ce înseamnă că orice element are date cât și o serie de metode prin care putem manipula aceste date. Python are la baza o implementare în C și Python unde sunt definite tipurile de date fundamentale. Aceste tipuri de date formează așa numitul data model<sup>9</sup>. Toate celelalte funcționalități și biblioteci sunt construite plecând de la acest model de date. Prin urmare, putem considera că într-un fel, limbajul este un API peste aceste elemente de bază.

Există mai multe implementări ale limbajului, cea mai des întâlnită fiind CPython. Aceasta versiune fiind și cea care conține cele mai noi funcționalități ale limbajului. În afară de această implementare mai există Jython (Python implementat în Java), Python for .NET (o implementare CPython care dispune de bibliotecile .NET), IronPython (o implementare care compilează codul Python în .NET) și PyPy (o implementare a limbajului scrisă în Python). Puteți afla mai multe detalii consultând documentația oficială sau accesând referința de mai jos<sup>10</sup>.

---

<sup>9</sup> Pentru mai multe informații despre Python Data Model, consultați documentația oficială <https://docs.python.org/3/reference/datamodel.html>

<sup>10</sup> Implementări alternative ale limbajului Python: <https://docs.python.org/3/reference/introduction.html#implementations>

Cu toate că limbajul este implementat în C, Python este un limbaj dinamic ceea ce l-a propulsat în topul multor domenii, în special în zona de analiză de date, știința datelor, sau învățarea automată. Poate vă întrebați în acest moment de ce se bucură Python de aceasta popularitate. Sincer să fiu, nu va pot oferi un răspuns concret. O posibilă explicație poate fi ușurința cu care putem scrie cod și cu care putem să ne concentrăm pe soluția problemei. Aceste caracteristici fac din Python un limbaj foarte eficient din punctul de vedere al timpului petrecut dezvoltând soluții. O altă explicație ar putea fi bariera scăzută de învățare. Python are o sintaxă accesibilă, care permite oricui să realizeze programe funcționale, fără a avea nevoie de cunoștințe avansate de informatică. Însă, acestea sunt doar supoziții. Cum vă spuneam și mai devreme, nu știu cu precizie de ce Python a căpătat o asemenea amploare în lumea învățării automate. Însă, un lucru este cert, acest limbaj s-a „stabilit” ca fiind unul dintre limbajele fundamentale pentru oricine vrea să dezvolte sisteme inteligente de procesare a datelor sau algoritmi de învățare automată.

Mai devreme vă spuneam că Python este un limbaj orientat pe obiect (OOP). Cu toate acestea, în Python putem folosi mai multe moduri de programare, inclusiv programare imperativă, funcțională, procedurală, și programarea orientată pe obiect. Detaliile fiecărui tip de programare depășesc scopul acestei lucrări, dar dacă sunteți interesați puteți afla mai multe consultând referințele (Ramalho, 2015).

În încheiere, țineți cont că Python este un limbaj foarte permisiv, care nu are nici un fel de limitări în privința codului pe care îl scrieți. Cu alte cuvinte, limbajul nu impune o structură anume sau un anumit mod de gândire. Din acest motiv, poate fi un limbaj ușor de învățat pentru începători, dar foarte greu de utilizat eficient. În Python, programatorul este cel care trebuie să decidă cum să își structureze și să își organizeze codul în funcție de aplicația pe care o dezvoltă. De asemenea, tot programatorul este cel responsabil de a se asigura că a scris un cod funcțional și că nu există erori de sintaxă. Acest lucru se datorează faptului că limbajul nu este compilat, prin urmare, interpretorul nu va ști dacă o anumită expresie este greșită decât în momentul în care aceasta este evaluată. Din fericire, există soluții prin care să putem verifica sintaxa codului pe care îl scriem, de exemplu, un alt program numit lint sau linter<sup>11</sup> ce analizează codul scris de noi și găsește erori de programare (sintaxă), bug-uri, sau alte elemente suspicioase.

În plus, versiunile recente de Python implementează un sistem de „typing” cu ajutorul căruia putem specifica tipul variabilelor. Însă, introducerea acestui concept nu înseamnă că putem să identificăm când folosim greșit anumite elemente (variabile, funcții, obiecte) din codul nostru deoarece interpretorul nu este „conștient” de tipuri. Totuși, sistemul de „typing” ne ajută ca si dezvoltatori să specificăm ce se așteaptă o funcție să primească ca și parametri (sau ce returnează aceasta funcție) (vezi secțiunile viitoare pentru mai multe detalii).

---

<sup>11</sup> Gasiți aici mai multe informații despre modul de funcționare al unui linter: <https://www.testim.io/blog/what-is-a-linter-heres-a-definition-and-quick-start-guide/>, <https://sourcelevel.io/blog/what-is-a-linter-and-why-your-team-should-use-it>

În concluzie, Python este un limbaj „high-level”, orientat pe obiect, dinamic și interpretat. Cu ajutorul acestui limbaj putem instrui calculatorul să efectueze orice sarcină. Mai mult, Python are o sintaxă intuitivă, ușor de învățat și o serie de pachete și biblioteci cu ajutorul cărora putem realiza aplicații complexe într-un timp relativ scurt.

### 3.1.1. Primul program în Python

Ok, am tot lăudat limbajul, dar cum putem realiza un program în Python? Blocul de cod de mai jos conține cel mai simplu program implementat în Python prin care instruiem calculatorul să afișeze pe ecran șirul de caractere „Hello, World!”.

```
print("Hello, World!")
```

Exemplul de cod 1. Un program care afișează pe ecran șirul de caractere „Hello, World!”

Dacă doriți să implementați acest prim program în Python, începeți prin a deschide editorul de text sau IDE<sup>12</sup>-ul preferat și copiați linia de cod din Exemplul 1 într-un fișier nou cu extensia „.py”, spre exemplu, „*hello\_world.py*”. Ulterior, pentru a putea rula acest program aveți nevoie de un interpretor de Python. Există mai multe variante de interpretoare dintre care puteți alege, cel mai folosit fiind CPython<sup>13</sup> pe care îl găsiți aici: <https://www.python.org/downloads/>

Interpretorul este un program creat special pentru a „traduce” codul scris folosind limbajul Python într-un limbaj (cod mașină) pe care calculatorul îl poate rula. Cu alte cuvinte, acest interpretor trimite instrucțiunile scrise de voi către calculator într-un limbaj pe care acesta îl înțelege astfel încât să le poată executa. Odată instalat interpretorul, puteți rula programul scris în fișierul „*hello\_world.py*” pe care l-ați creat mai devreme. Găsiți mai jos pașii necesari pentru a rula acest program (găsiți un exemplu în figura 4).

1. Deschideți command prompt-ul (daca folosiți sistemul de operare Windows) sau terminalul (daca folosiți un sistem de operare Unix).
2. Navigați în folderul ce conține fișierul „*hello\_world.py*”
3. Introduceți următoarea comandă în terminal, *python hello\_word.py*, și apăsați tasta Enter.

În urma efectuării pașilor de mai sus ar trebui să vedeți afișat în terminal textul *Hello, World!* Felicitări! Tocmai ce ați reușit să scrieți și să rulați primul vostru program în Python!

---

<sup>12</sup> Integrated Development Environment (Mediul de dezvoltare integrat):  
<https://www.redhat.com/en/topics/middleware/what-is-ide>

<sup>13</sup> Puteți afla mai multe detalii despre CPython și alternativele sale aici:  
<https://docs.python.org/3/reference/introduction.html#implementations>

```
(base) Andreis-MacBook-Pro-2:Source luchicla$ python hello_world.py
Hello, World!
(base) Andreis-MacBook-Pro-2:Source luchicla$ █
```

Figura 4. Rezultatul rulării codului hello\_world.py

Sintaxa limbajului este intuitivă. Creatorul limbajului și-a dorit ca aceasta să fie ușor de citit pentru începători astfel încât să elimine multe simboluri folosite de alte limbaje de programare precum „;” pentru a semnaliza finalul unei linii de cod sau „{, și ,}” pentru a semnaliza un set de instrucțiuni. Mai mult, în Python blocurile de cod din interiorul unei structuri sunt delimitate cu ajutorul unui tab.

În cele ce urmează vom explora câteva noțiuni fundamentale ale limbajului pentru a ne familiariza cu acesta; începem cu tipurile fundamentale de date și terminăm prin a vedea cum definim propriile tipuri de date și obiecte.

### 3.1.2. Tipuri de date fundamentale

Elementul fundamental în Python este un obiect. Un obiect reprezintă o instanță concretă a unui tipar predefinit. Cu ajutorul acestor tipare specificăm o colecție de date și metode pentru accesul, modificarea, sau transformarea acestor date. Calculatorul le salvează în memorie sub o singură etichetă. Ulterior, putem folosi în programele pe care le dezvoltăm mai multe instanțe ale acestor tipare, particularizate cu date specifice. În Secțiunea 3.1.5 vom vedea cum putem crea aceste tipare și cum le putem folosi pentru a realiza programe complexe.

În Python, un obiect are următoarele caracteristici *identitate*, *tip* și *valoare*. Identitatea unui obiect este stabilită de către interpretor în momentul în care acesta este creat. În plus, aceasta rămâne aceeași pe toată durata execuției acelui program. Putem să ne gândim la această identitate ca la adresa de memorie (locația din memorie) unde a fost inserat acel obiect. Putem folosi identitatea unor obiecte când vrem să stabilim dacă acestea sunt aceleași, utilizând metoda *id()*, pentru a extrage identitatea fiecărui obiect.

Tipul obiectului determină ce operații putem efectua cu acel obiect și limitează modul în care putem folosi obiectul. Mai mult, tipul unui obiect specifică și ce valori putem atribui acestuia. De exemplu, un obiect de tip număr întreg nu poate avea ca și valoare un șir de caractere și viceversa. Tipul unui obiect este determinat la runtime, atunci când interpretorul întâlnește pentru prima dată linia prin care acel obiect este creat. În Python există mai multe metode prin care putem evalua tipul unui obiect, una dintre acestea fiind metoda *type()*.

Tipurile fundamentale de date includ: *None*, *NotImplemented*, *Ellipsis*, *Numbers (int, float)*, *Sequences (lists, tuples, arrays)*, *Sets*, *Mappings (dictionaries)*, *Callables (functions)*, *Modules*, *Classes*, *Instances of classes*, *I/O objects*, și *user-defined types*<sup>14</sup>.

Valoarea unui obiect o reprezintă datele aflate în memorie la adresa unde a fost stocat obiectul. În Python există două tipuri de valori: imutabile și mutabile. Obiectele care au valori imutabile reprezintă eticheta unei locații de memorie care nu poate fi modificată în mod direct. Dacă dorim să modificăm valoarea unui obiect imutabil, interpretorul va atribui noii valori o nouă zonă de memorie pentru stocarea acesteia, ștergând zona de memorie unde se regăsea valoarea anterioară. În contrast, obiectele care au valori mutabile pot modifica aceeași zonă de memorie. Cu alte cuvinte, dacă modificăm valoarea unui obiect mutabil vom modifica valoarea existentă în memorie la aceeași adresă ca atunci când acel obiect a fost creat. Aceasta distincție dintre valori mutabile și imutabile poate conduce la apariția unei situații interesante când creăm un obiect imutabil care conține o referință către un obiect mutabil. În acest caz, putem modifica valoarea conținută de obiectului imutabil, dar nu îl putem modifica pe acesta. Veți vedea un exemplu al acestei situații când vom discuta despre liste și tuple, două tipuri fundamentale în Python, unul mutabil și celălalt imutabil.

În cele ce urmează vom vedea cum putem instanția diferite obiecte din tipurile fundamentale de date. Pentru mai multe detalii despre aceste tipuri de date puteți consulta referința (Ramalho, 2015) sau standardul oficial al limbajului.

```
# Initializarea unei variabile nedefinite
valoare_nedefinita = None

# Initializarea unor variabile de tip numeric
numar_intreg = 1
numar_zecimal = 10.24

# Initializarea unor variabile de tip boolean
variabila_booleana_adevarata = True
variabila_booleana_falsa = False

# Initializarea unor variabile de tip string
sir_de_caractere_1 = "Acesta este un sir de caractere."
sir_de_caractere_2 = 'Acesta este alt sir de caractere.'
sir_de_caractere_3 = """Acesta este un sir de caractere care
ocupa mai multe randuri."""
sir_de_caractere_4 = f"Acesta este un sir de caractere interpolat care contine valoarea variabilei
'numar_intreg' = {numar_intreg}."
sir_de_caractere_5 = f'Acesta este un sir de caractere interpolat care contine valoarea variabilei
"numar_intreg" = {numar_intreg}.'
sir_de_caractere_6 = f"""Acesta este un sir de caractere interpolat care contine valoarea variabilei
'numar_intreg' = {numar_intreg}."""
```

Exemplul de cod 2. Inițializarea unor variabile de diferite tipuri.

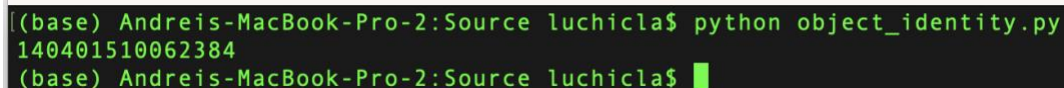
---

<sup>14</sup> Găsiți standardul la zi al limbajului folosind următorul link  
<https://docs.python.org/3/reference/datamodel.html>

Dacă vrem să aflăm identitatea unui obiect, putem folosi metoda `id()` căreia îi oferim ca și parametru numele variabilei a cărei identitate dorim să o aflăm (vezi exemplele de mai jos). Când rulați liniile de cod din exemplul de mai jos, veți vedea afișat pe ecran un număr similar cu cel din figura 5.

```
# Id-ul unui obiect
numar_intreg_id = id(numar_intreg)
print(numar_intreg_id)
```

Exemplul de cod 3. Inițializarea unor variabile de diferite tipuri.



```
(base) Andreis-MacBook-Pro-2:Source luchicla$ python object_identity.py
140401510062384
(base) Andreis-MacBook-Pro-2:Source luchicla$
```

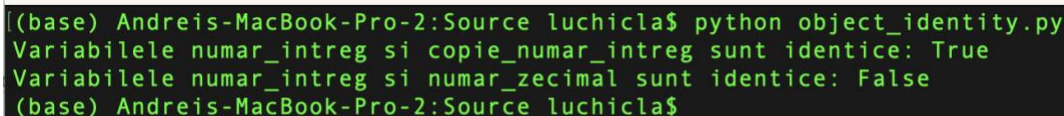
Figura 5. Exemplul identității unui obiect.

Mai mult, dacă dorim să comparăm două obiecte putem să aflăm identitatea acestora și ulterior să o comparăm, similar cu exemplul de cod 4. În figura 6 regăsiți rezultatul execuției liniilor de cod din acest exemplu. Din câte puteți observa, variabilele `numar_intreg` și `copie_numar_intreg` reprezintă același obiect, ele fiind o etichetă pentru aceeași valoare din memorie. În schimb, `numar_intreg` și `numar_zecimal` au identități diferite, reprezentând valori diferite în memoria calculatorului. În mod similar putem afla sau compara tipurile unor obiecte înlocuind metoda `id()` cu metoda `type()`.

```
# Compararea identitatii obiectelor
copie_numar_intreg = numar_intreg
obiecte_identice = id(numar_intreg) == id(copie_numar_intreg)
print(f"Variabilele numar_intreg si copie_numar_intreg sunt identice: {obiecte_identice}")

obiecte_diferite = id(numar_intreg) == id(numar_zecimal)
print(f"Variabilele numar_intreg si numar_zecimal sunt identice: {obiecte_diferite}")
```

Exemplul de cod 4. Compararea identității obiectelor.



```
(base) Andreis-MacBook-Pro-2:Source luchicla$ python object_identity.py
Variabilele numar_intreg si copie_numar_intreg sunt identice: True
Variabilele numar_intreg si numar_zecimal sunt identice: False
(base) Andreis-MacBook-Pro-2:Source luchicla$
```

Figura 6. Compararea identității unor obiecte.

Tipurile de date pe care le-am văzut până acum nu pot stoca decât o singură valoare. În practică există multe situații în care vrem să stocăm colecții de date cu o anumită structură. De exemplu, un tabel cu date unde avem mai multe rânduri și coloane. Dacă ar fi să reprezentăm acest tabel folosind doar tipurile pe care le-am parcurs până acum, ar însemna să creăm manual câte o



variabilă pentru fiecare celulă din tabel ceea ce nu ne-ar fi de mare ajutor atunci când vrem să modificăm valoarea unei celule sau când vrem să calculăm diferite metrici pentru una sau mai multe coloane sau pentru unul sau mai multe rânduri. Din fericire însă, avem în Python mai multe tipuri de date pe care le-am putea folosi pentru a reprezenta colecții de date.

Primul tip de date folosit pentru a manipula colecții de date îl reprezintă secvențele. Acestea sunt împărțite în secvențe mutabile și imutabile. În cazul secvențelor mutabile putem modifica valoarea unui element din colecția creată inițial. În schimb, secvențele imutabile nu permit modificarea valorii unui element după ce acestea au fost inițializate.

Listele sunt cele mai întâlnite secvențe mutabile. Ele sunt de asemenea și secvențe ordonate. Listele sunt folosite pentru a stoca o colecție de elemente de același tip sau de diferite tipuri într-o manieră secvențială în memorie. Elementele listei pot fi accesate folosind indexul (poziția) elementului din lista. Primul element dintr-o lista are indexul 0 (Python indexează elementele unei secvențe începând cu 0) (Figura 7). Pentru a accesa un element dintr-o listă, ne folosim de numele variabilei și între paranteze pătrate „[]” specificăm poziția elementului pe care dorim să îl accesăm. Pe lângă funcționalitatea de a stoca o colecție de elemente, o lista, fiind un obiect, are și o serie de metode predefinite cu ajutorul cărora putem face diferite operații, precum adăugarea unor elemente noi (*append*), ștergerea unor elemente existente (*remove* sau *pop*) sau sortarea în ordine crescătoare sau descrescătoare a elementelor (*sort*) (vedeți și exemplul de cod 5 împreună cu rezultatul execuției fiecărei secvențe de cod). În afară de operațiuni pentru modificarea unei liste, Python ne oferă și un cuvânt cheie special, *in*, pe care îl putem folosi să verificăm dacă o anumită valoare există într-o listă. Pentru mai multe informații despre implementarea listelor în Python sau alte operații pe care le puteți face cu ele, consultați (Ramalho, 2015) sau documentația oficială a limbajului<sup>15</sup>

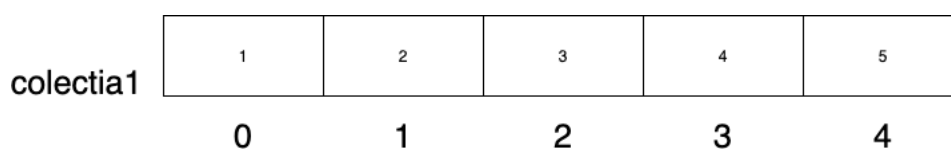


Figura 7. O lista cu 5 elemente. Observați cum elementele sunt stocate secvențial în memorie și indexate începând cu indexul 0.

```
# Liste
# o lista cu elemente de tip int
colectia1 = [1, 2, 3, 4, 5]

# o lista cu elemente de tip string
colectia2 = ['Aceasta', 'este', 'o', 'colectie']

# o lista cu elemente de mai multe tipuri, int, string, si chiar alte liste
colectia3 = ['a', 1, 2, [3, 4, 'b'], ['c', 5], 6]

# initializarea unei liste folosind o alta colectie
colectia4 = list(colectia1)
```

<sup>15</sup> <https://docs.python.org/3/tutorial/datastructures.html>

```

print(colectia1)
>> [1, 2, 3, 4, 5]

print(colectia2)
>> ['Aceasta', 'este', 'o', 'colectie']

print(colectia3)
>> ['a', 1, 2, [3, 4, 'b'], ['c', 5], 6]

print(colectia4)
>> [1, 2, 3, 4, 5]

# Modificarea valorii unui element al listei
colectia1[0] = 'Valoare noua'
print(colectia1)
>> ['Valoare noua', 2, 3, 4, 5, 'Element adaugat la sfarsitul colectiei']

# Accesarea valorilor dintr-o lista
print(colectia1[2])
>> 3

# Exemple de operatii premise cu liste
# Sortarea
colectia4.sort(reverse=True)
print(colectia4)
>> [5, 4, 3, 2, 1]

# Adaugarea unui element nou
colectia1.append("Element adaugat la sfarsitul colectiei")
print(colectia1)
>> ['Valoare noua', 2, 3, 4, 5, 'Element adaugat la sfarsitul colectiei']

# Verificarea existentei unei valori intr-o lista
"Valoare noua" in colectia1
>> True

```

Exemplul de cod 5. Inițializarea listelor și operațiuni cu liste. „>>” reprezintă rezultatul afișat pe ecran în urma execuției secvenței de cod.

Tuplurile (tuples) sunt o altă structură de date cu ajutorul căreia putem stoca o colecție în memorie. Spre deosebire de liste, tuplurile sunt secvențe imutabile. Cu alte cuvinte, o dată creat un tuplu, nu putem modifica valorile elementelor acestuia. Există însă o excepție, și anume, atunci când valoarea elementului este mutabilă, precum o listă. Dacă avem un tuplu ce conține liste, putem să modificăm valorile conținute de acele liste (vezi exemplele de cod de mai jos). În afară de această distincție, tuplurile pot fi folosite în același mod ca și listele. Pentru mai multe detalii despre implementarea acestora și operațiile permise, puteți consulta (Ramalho, 2015) sau documentația oficială a limbajului<sup>16</sup>.

---

<sup>16</sup> <https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences>



```

# Tuples
# Initializarea unui tuplu
tuple1 = (1, 2, 3, 3, 3, 3, 4, 5)
print(tuple1)
>> (1, 2, 3, 3, 3, 3, 4, 5)

# Initializarea unui tuplu dintr-un alt tuplu
tuple2 = tuple(tuple1)
print(tuple1)
>> (1, 2, 3, 3, 3, 3, 4, 5)

# Intializarea unui tuplu dintr-o colectie
tuple3 = tuple(colectia1)
print(tuple3)
>> ('Valoare noua', 2, 3, 4, 5, 'Element adaugat la sfarsitul colectiei')

elemente = tuple1.count(3)
print(f"Numarul de elemente egale cu 3 din tuple1: {elemente}")
>> Numarul de elemente egale cu 3 din tuple1: 4

indexul_unui_element = tuple2.index(3)
print(f"Indexul elementului cu valoarea 3 este: {indexul_unui_element}.")
>> Indexul elementului cu valoarea 3 este: 2.

# Accesarea unei valori din tuplu
print(tuple3[1])
>> 2

# Accesarea mai multor valori din tuplu
print(tuple3[1:3])
>> (2, 3)

```

Exemplul de cod 6. Inițializarea tuplurilor și operațiuni cu tupluri. „>>” reprezintă rezultatul afișat pe ecran în urma execuției secvenței de cod.

Python nu ar fi un limbaj complet dacă nu ne-ar oferi opțiunea de stoca elementele într-o matrice. La fel ca în restul limbajelor de programare, matricele sunt folosite pentru a crea colecții uni sau multidimensionale de date de același tip. În Python folosim matricele similar cu listele, ele fiind recomandate atunci când știm ce tipuri de date va conține colecția și mai ales dacă acele date sunt de același tip deoarece putem reduce memoria necesară pentru stocarea acelor date. În secțiunea 3.1.6 vom vedea mai multe detalii despre diferite biblioteci specializate în procesarea matricelor.

Seturile<sup>17</sup> sunt ultimul tip de secvențe pe care îl vom discuta. Seturile sunt tipuri de date imutabile care conțin doar elementele unice dintr-o secvență într-o ordine aleatorie. Ele sunt echivalentul mulțimilor cu elemente unice din matematica. Chiar dacă sunt asemănătoare cu restul secvențelor, seturile nu ne permit să extragem elemente individuale deoarece nu sunt ordonate. Cu ajutorul seturilor putem efectua operații cu diferite mulțimi, precum intersecții, uniuni și diferențe. În exemplul de cod 7 puteți observa cum inițializăm și cum lucrăm cu seturi.

---

<sup>17</sup> <https://docs.python.org/3/tutorial/datastructures.html#sets>

```

# Seturi
# Initializarea unui set
setul1 = {1, 2, 3, 4}
print(setul1)
>> {1, 2, 3, 4}

# Initializarea unui set dintr-o lista
setul2 = set(tuple1)
print(setul2)
>> {1, 2, 3, 4, 5}

# Intersectia a doua seturi
intersectie = setul1.intersection(setul2)
print(intersectie)
>> {1, 2, 3, 4}

# Uniunea a doua seturi
uniune = setul1.union(setul2)
print(uniune)
>> {1, 2, 3, 4, 5}

# Diferenta dintre doua seturi
diferenta = setul2.difference(setul1)
print(diferenta)
>> {5}

# Adaugarea unei valori in set
setul2.add(9)
print(setul2)
>> {1, 2, 3, 4, 5, 9}

# Eliminarea unei valori din set
setul2.remove(3)
print(setul2)
>> {1, 2, 4, 5, 9}

```

Exemplul de cod 7. Inițializarea seturilor și operațiuni cu seturi. „>>” reprezintă rezultatul afișat pe ecran în urma execuției secvenței de cod.

Toate structurile de date prezentate mai sus (liste, tupluri și seturi) fac parte dintr-un tip de date numit „iterable” deoarece pot fi parcurse element cu element într-o buclă repetitivă (mai multe detalii despre acestea la sfârșitul acestei secțiuni). În plus, elementele dintr-o listă și tuplu pot fi accesate pe porțiuni pentru a extrage sau actualiza (după caz) un număr mai mare de element. Pentru a extrage o secțiune dintr-o listă sau tuplu, specificăm, între paranteze drepte, elementul de start și elementul de final separate prin „:”. Mai mult, putem specifica și pasul cu care să extragă acea secțiune. În schimb, seturile nu pot fi accesate prin această metodă deoarece nu permit accesul la elementele individuale.

În afara secvențelor, Python ne oferă posibilitatea de a stoca date cu o structură arbitrară prin maparea acestora la niște chei. Aceste tipuri de date poartă denumirea generică de mapări. Cea mai des întâlnită formă de mapare este dicționarul sau *dict*. Acest tip de date ne permite să

stocăm informațiile folosind o structura de (cheie, valoare), unde cheia poate fi orice tip de date „hashable”<sup>18</sup>, iar valorile pot fi reprezentate de orice tip de date, chiar și alte dicționare sau tipuri de date definite de către programatori. Fiind un obiect, un dicționar are la rândul lui o serie de metode pe care le putem folosi să accesăm elementele acestuia, precum *keys()* care returnează cheile dicționarului, *values()* care returnează valorile fiecărei chei din dicționar sau *items()* care returnează o colecție de tuple-uri sub forma (*cheie, valoare*) pe care le putem utiliza pentru a parcurge datele conținute în dicționar. În secvențele de cod de mai jos puteți vedea cum creăm un dicționar și cum îl folosim.

```
# Dictionare
# Initializarea unui dictionar
dict1 = {
    "cheia1": "valoarea1",
    "cheia2": 1,
    "cheia3": [1, 2, 3],
    "cheia4": (1, 2, 3),
    "cheia5": {
        "cheia6": 1,
        "cheia7": 3
    }
}

# Accesare elementelor unui dictionar
print(dict1["cheia1"])
print(dict1["cheia5"])
print(dict1["cheia5"]["cheia6"])
>> valoarea1
>> {'cheia6': 1, 'cheia7': 3}
>> 1

# Modificare valorilor unui dictionar
dict1["cheia1"] = "noua valoare 1"
print(dict1)
>> {'cheia1': 'noua valoare 1', 'cheia2': 1, 'cheia3': [1, 2, 3], 'cheia4': (1, 2, 3), 'cheia5': {'cheia6': 1, 'cheia7': 3}}

dict1["cheia8"] = "valoarea unei chei noi"
print(dict1)
>> {'cheia1': 'noua valoare 1', 'cheia2': 1, 'cheia3': [1, 2, 3], 'cheia4': (1, 2, 3), 'cheia5': {'cheia6': 1, 'cheia7': 3},
'cheia8': 'valoarea unei chei noi'}

# Accesare cheilor dintr-un dictionar
print(list(dict1.keys()))
>> ['cheia1', 'cheia2', 'cheia3', 'cheia4', 'cheia5', 'cheia8']

# Accesarea valorilor dintr-un dictionar
print(list(dict1.values()))
>> ['noua valoare 1', 1, [1, 2, 3], (1, 2, 3), {'cheia6': 1, 'cheia7': 3}, 'valoarea unei chei noi']

# Accesarea elementor unui dictionar
print(list(dict1.items()))
>> [('cheia1', 'noua valoare 1'), ('cheia2', 1), ('cheia3', [1, 2, 3]), ('cheia4', (1, 2, 3)), ('cheia5', {'cheia6': 1,
'cheia7': 3}), ('cheia8', 'valoarea unei chei noi')]
```

---

<sup>18</sup> <https://www.pythonmorsels.com/what-are-hashable-objects/>

Exemplul de cod 8. Inițializarea dicționarelor și operațiuni cu dicționare. „>>” reprezintă rezultatul afișat pe ecran în urma execuției secvenței de cod.

### 3.1.3. Expresii și structuri de control

Programarea nu ar fi foarte folositoare dacă tot ce am putea instrui un calculator să facă ar fi să stocheze diferite date în anumite structuri și ulterior să modificăm aceste valori. Prin urmare, creatorii limbajelor de programare ne oferă o suită de unelte cu ajutorul cărora să putem realiza expresii a căror valoare să fie evaluată, structuri de control al execuției programului cu ajutorul cărora să putem specifica seturi de instrucțiuni ce urmează să fie executate dacă anumite condiții sunt îndeplinite sau cu ajutorul cărora să executăm mai multe instrucțiuni în mod repetat. În această secțiune vom parcurge elementele din Python cu ajutorul cărora putem realiza toate aceste lucruri. Dar, înainte să începem trebuie să discutăm despre operatori. Operatorii sunt cuvinte cheie cu care putem specifica interpretorului o operație predefinită. De exemplu, dacă dorim să efectuăm calcule aritmetice, ne putem folosi de operatorii aritmetici, „+”, „-”, „/”, „\*” (adunare, scădere, împărțire, și respectiv înmulțire). Aceștia sunt operatori binari (au nevoie de două valori sau variabile asupra cărora să acționeze, una la stânga și cealaltă la dreapta). Există însă o excepție, „-”, care poate fi folosit atât ca operator binar cât și să specificăm o valoare negativă. Mai mult, fiecare tip de date are o implementare proprie pentru a calcula rezultatul unei operații aritmetice.

Pe lângă operatorii aritmetici, Python ne oferă și o serie de operatori cu care să construim expresii logice. Acești operatori ne permit să construim expresii logice complexe. Cei mai cunoscuți operatori logici sunt AND (*and*), OR (*or*) și (*not*). Mai mult, există situații când dorim să efectuăm diferite comparații, de exemplu, să verificăm dacă două variabile sunt egale (==) sau diferite (!=) sau dacă valoarea unei variabile este mai mică sau mai mare decât valoarea unei alte variabile.

În concluzie, Python ne oferă o selecție de operatori cu ajutorul cărora putem construi expresii aritmetice sau logice. În tabelul 1 găsiți cei mai întâlniți operatori și o scurtă explicație a acestora. Este important să observați că valoarea expresiei diferă în funcție de tipul operatorului (logic sau aritmetic). În cazul expresiilor aritmetice, rezultatul este un obiect de același tip cu operandii și cu valoarea egală cu rezultatul operațiilor efectuate. În schimb, expresiile logice rezultă într-un obiect de tipul bool cu valoarea egală cu valoarea de adevăr a expresiei respective (True sau False).

Operator	Tip	Cuvânt cheie	Note
Adunare	Aritmetic	+	Operator binar, implementat pentru numere și șiruri de caractere
Scădere	Aritmetic	-	Operator binar sau unar atunci când

			reprezintă o valoare negativă, implementat pentru numere
Înmulțire	Aritmetic	*	Operator binar, implementat pentru numere, șiruri de caractere și secvențe
Împărțire	Aritmetic	/	Operator binar, implementat pentru numere
And	Logic	and	Operator binar
Or	Logic	or	Operator binar
Not	Logic	not	Operator unar, reprezintă negația unei expresii
Mai mic	Logic	<	Operator binar, folosit pentru compararea valorilor a doua variabile
Mai mic sau egal	Logic	<=	Operator binar, folosit pentru compararea valorilor a doua variabile
Mai mare	Logic	>	Operator binar, folosit pentru compararea valorilor a doua variabile
Mai mare sau egal	Logic	>=	Operator binar, folosit pentru compararea valorilor a doua variabile
Egal	Logic	==	Operator binar, folosit pentru compararea valorilor a doua variabile
Diferit	Logic	!=	Operator binar, folosit pentru compararea valorilor a doua variabile
-	Logic	is	Operator binar, folosit pentru a stabili dacă două variabile se referă la același obiect
Conține	Logic	in	Operator binar, folosit pentru a stabili dacă o secvență conține o anumită valoare

### Tabelul 1. Lista cu cei mai întâlniți operatori în Python

Folosind expresii putem instrui calculatorul să efectueze calcule. În cele ce urmează, vom vedea cum putem integra aceste expresii în diferite structuri de control pentru a realiza algoritmi de calcul. În Python structurile de control disponibile sunt: IF-ELSE, FOR, WHILE.

Structura IF-ELSE ne permite să realizăm mai multe ramuri în programele pe care le dezvoltăm astfel încât să putem executa diferite instrucțiuni în funcție de anumite condiții. Sintaxa structurii IF-ELSE este destul de intuitivă (vezi exemplele de cod de mai jos). Interpretorul începe execuția unei structuri IF-ELSE prin a evalua condiția de pe ramura IF. Dacă această condiție este evaluată ca fiind adevărată (True) atunci codul din interiorul ramurii IF este executat. La final, interpretorul sare peste ramura ELSE și revine la următoarea linie de cod. Dacă evaluarea condiției rezultă într-o valoare False, atunci interpretorul execută instrucțiunile din interiorul ramurii ELSE și revine la următoarea linie de cod. Mai mult, putem avea mai multe ramuri IF-ELSE imbricate, iar ramura ELSE este opțională. În cazul în care avem nevoie să evaluăm mai multe condiții, putem extinde structura IF-ELSE cu ramuri adiționale ELIF care să specifice câte o condiție și un set de instrucțiuni pe care dorim să le executăm dacă o anumită condiție este evaluată ca True. În cazul unei structuri IF-ELIF-ELSE execuția este secvențială, evaluarea condițiilor făcând-se de sus în jos, începând cu ramura IF, trecând prin fiecare ramura ELIF și terminând cu ELSE.

```
# IF-ELIF-ELSE
a = 10
b = 20
if a > b:
    print(f"a = {a} is greater than b = {b}")
elif a < b:
    print(f"a = {a} is less than b = {b}")
else:
    print(f"a = {a} is equal to b = {b}")
>> a = 10 is less than b = 20
```

### Exemplul de cod 9. Structura IF-ELIF-ELSE

Structura FOR ne permite să executăm un set de instrucțiuni de un număr prestabilit de ori. Cu ajutorul acestei structuri putem să parcurgem elementele unei secvențe și să executăm pentru fiecare element aceleași instrucțiuni. Exemplul de cod 10 prezintă sintaxa structurii FOR pentru parcurgerea în ordine a elementelor unei liste.

```
# for loops
colectie = ["ELEMENT1", "ELEMENT2", "ELEMENT3"]
for element in colectie:
    element = element.lower()
    print(element)
>> element1
>> element2
>> element3
```

### Exemplul de cod 10. Structura FOR

Structura WHILE ne permite să executăm un set de instrucțiuni cât timp o condiție este adevărată. În cazul acestei structuri este important ca la un moment dat condiția să devină falsă. În caz contrar, programul va intra într-un ciclu infinit și va executa la nesfârșit instrucțiunile din interiorul structurii. Exemplul de cod 11 prezintă sintaxa structurii WHILE pentru parcurgerea în ordine inversă a elementelor unei liste.

```
# while loops
colectie = ["ELEMENT1", "ELEMENT2", "ELEMENT3"]
n = len(colectie) - 1
while n >= 0:
    element = colectie[n]
    element = element.lower()
    print(element)
    n = n - 1
>> element3
>> element2
>> element1
```

Exemplul de cod 11. Structura WHILE

### 3.1.4. Introducere în funcții

Funcțiile sunt unele dintre cele mai folosite structuri pentru a grupa un set de instrucțiuni pe care dorim să le refolosim în mai multe zone ale programelor noastre.

Pentru a defini o funcție, mai întâi „înștiințăm” interpretorul ca ceea ce urmează face parte din corpul unei funcții cu ajutorul cuvântului cheie *def* urmat de numele funcției și între paranteze parametrii acelei funcții. Terminăm aceasta declarație cu „: „, (la fel ca în cazul structurilor IF-ELSE, FOR și WHILE).

O funcție poate avea zero, unul, sau mai mulți parametri. Aceștia reprezintă datele de intrare necesare pentru executarea cu succes a tuturor instrucțiunilor din corpul funcției. Mai mult, acești parametri pot fi obligatorii sau opționali și pot avea o valoare predefinită.

Corpul funcției poate conține una sau mai multe instrucțiuni și opțional poate returna o valoare. Instrucțiunile pot fi alcătuite cu ajutorul tuturor structurilor pe care le-am parcurs până acum. Indiferent de complexitatea instrucțiunilor din interiorul funcției, acestea trebuie aliniate cu un tab la dreapta (vezi exemplele de mai jos) la fel ca orice alt set de instrucțiuni din interiorul unei structuri.

Nu este suficient doar să definim o funcție. Pentru a ne putea folosi de aceasta, trebuie să o apelăm. Apelarea unei funcții se realizează cu ajutorul numelui funcției și, opțional, între paranteze valorile parametrilor de intrare. Este important de observat că pentru funcțiile cu parametri opționali nu este nevoie să specificăm în mod explicit valoarea acelor parametri. Interpretorul va căuta o valoare și în cazul în care nu o găsește va folosi valoarea predefinită.

În exemplul de cod 12 vom vedea sintaxa prin care definim o funcție și câteva exemple de funcții cu și fără parametri, funcții cu parametri poziționali și funcții cu parametri definiți.

```
# Functii
# Definirea unei functii
def sum(a=10, b=10):
    # calculeaza suma celor doua numere, a si b
    c = a + b
    # returneaza valoarea sumei
    return c

def is_odd(a):
    # verifica daca restul impartirii lui a la 2 este zero
    # daca restul este zero, numarul este par.
    if a % 2 == 0:
        return False
    else:
        return True

def factorial(N):
    # verifica daca N este un numar intreg si arunca o eroare in caz contrar
    if not isinstance(N, int):
        raise ValueError("Valoarea lui N este incorecta. Incercati din nou folosind un numar intreg.")

    # verifica si arunca o eroare daca N <= 0
    if N <= 0:
        raise ValueError("Valoarea lui N este incorecta. Incercati din nou folosind un numar intreg mai mare decat 0.")

    # initializeaza valoarea produsului cu 1
    product = 1

    # pentru fiecare numar intre 1 si N, inmulteste produsul cu numarul respectiv
    for number in range(1, N+1):
        product = product * number

    # returneaza valoarea finala a produsului
    return product

# Apelarea functiilor
c = sum()
print(c)
>> 20

c = sum(1, 1)
print(c)
>> 2

x = 10
y = 20
z = sum(x, y)
print(z)
>> 30

odd = is_odd(x)
print(f"x = {x} este un numar impar: {odd}.")
```



```
>> x = 10 este un numar impar: False.
```

```
t = 10
p = factorial(t)
print(f"{t}! = {p}")
>> 10! = 3628800
```

### Exemplul de cod 12. Funcții.

În Python funcțiile sunt considerate obiecte de primă clasă. Aceasta particularitate ne permite să folosim funcții în mai multe situații, nu doar pentru a grupa o serie de instrucțiuni pe care le repetăm în mai multe componente ale programelor noastre.

În Python, funcțiile sunt obiecte de primă clasă, ceea ce înseamnă că pot fi tratate ca orice alt obiect din limbaj. Aceasta înseamnă că le puteți atribui variabilelor, le puteți transmite ca argumente altor funcții și le puteți returna ca valori din funcții.

Unul dintre principalele avantaje ale funcțiilor fiind obiecte de primă clasă este că permite funcții de ordin superior, care sunt funcții care iau alte funcții ca argumente sau returnează funcții ca ieșire. Acest lucru permite o mare flexibilitate și modularitate în codul dvs., deoarece puteți scrie funcții care pot fi personalizate și reutilizate în diferite contexte.

De exemplu, puteți defini o funcție care ia o altă funcție ca argument și o aplică unei liste de valori (vezi exemplul de cod 13). Puteți utiliza apoi această funcție pentru a aplica orice funcție doriți la o listă de valori, pur și simplu trecând funcția ca argument. Acest lucru poate fi foarte util pentru a îndepărta modelele comune și pentru a face codul mai reutilizabil.

În general, capacitatea de a trata funcțiile ca obiecte de primă clasă este o caracteristică puternică a Python care vă permite să scrieți cod mai flexibil și mai modular, folosind paradigma programării funcționale (Ramalho, 2015).

```
def factorial(N):
    # verifica daca N este un numar intreg si arunca o eroare in caz contrar
    if not isinstance(N, int):
        raise ValueError("Valoarea lui N este incorecta. Incercati din nou folosind un numar intreg.")

    # verifica si arunca o eroare daca N <= 0
    if N <= 0:
        raise ValueError("Valoarea lui N este incorecta. Incercati din nou folosind un numar intreg mai mare decat 0.")

    # initializeaza valoarea produsului cu 1
    product = 1

    # pentru fiecare numar intre 1 si N, inmulteste produsul cu numarul respectiv
    for number in range(1, N+1):
        product = product * number

    # returneaza valoarea finala a produsului
    return product
```

```
def aplica_functia(functie, valori):
    rezultat = []
    for x in valori:
        evaluare = functie(x)
        rezultat.append(evaluare)
    return rezultat

colectie = [1, 2, 3, 4]
valori_factoriale = aplica_functia(factorial, colectie)
print(valori_factoriale)
>> [1, 2, 6, 24]
```

#### Exemplul de cod 12. Funcții de ordin superior.

În afara de funcțiile de ordin superior, Python oferă opțiunea de a extinde comportamentul unei funcții fără să modificăm Python, un decorator este modifica corpul funcției. Pentru a realiza acest lucru, putem să implementăm o funcție care preia o altă funcție ca și parametru și îi extinde comportamentul. Aceste funcții se numesc decoratori și sunt folosite pentru a adăuga funcționalități suplimentare la funcțiile existente. Mai mult, decoratorii pot fi folositori pentru a abstractiza modelele comune și pentru a face codul mai modular și mai reutilizabil.

Pentru a folosi un decorator în Python, definim o funcție care ia o altă funcție ca argument și apoi utilizăm simbolul „@” urmat de numele funcției de decorator pentru a „decora” funcția originală. În exemplul de cod 13 am implementat un decorator simplu care adaugă un mesaj la valoarea returnată de funcția decorată.

```
# Decoratori
def schimba_rezultatul(functie):
    def wrapper():
        rezultat = functie()
        print(f"Am decorat rezultatul funcției. {rezultat}.")
    return wrapper

@schimba_rezultatul
def salutare():
    return "Salut!"

salutare()
>> Am decorat rezultatul funcției. Salut!.
```

#### Exemplul de cod 13. Un decorator simplu.

În acest exemplu, funcția *schimba\_rezultatul()* ia o funcție ca argument și definește o funcție *wrapper()* care adaugă textul „Am decorat rezultatul funcției.” la valoarea returnată de funcția originală. Funcția *salutare()* decorată este definită folosind decoratorul *@schimba\_rezultatul*, care înglobează funcția originală în funcția *wrapper()*.

Dacă această funcționalitate nu v-a surprins, pregătiți-vă pentru ce urmează! Decoratorii pot fi folosiți și pentru a transmite argumente suplimentare funcției decorate. De exemplu, putem

defini un decorator care preia un argument și îl folosește pentru a modifica rezultatul funcției decorate precum în exemplul de cod 14.

```
# Decorator cu parametri
def repeta(repetitii):
    def decorator(funcie):
        def wrapper(*args, **kwargs):
            for i in range(repetitii):
                funcie(*args, **kwargs)
            return wrapper
        return decorator

@repeta(repetitii=5)
def salutare(name):
    print(f'Salut, {name}!')

salutare("Victor")
>> Salut, Victor!
>> Salut, Victor!
>> Salut, Victor!
>> Salut, Victor!
>> Salut, Victor!
```

Exemplul de cod 14. Un decorator care acceptă parametri.

În acest exemplu, funcția *repeta()* ia un parametru *repetitii* și definește o funcție interioară *decorator()* care ia o funcție ca argument. Funcția *decorator()* definește o funcție *wrapper()* care apelează funcția decorată de mai multe ori folosind argumentul *repetiti*. Decoratorul *@repeata (repetitii=5)* include funcția *salutare()* în funcția *wrapper()*, determinând apelarea acesteia de cinci ori. În acest exemplu observați ca funcția *wrapper()* acceptă ca și parametri *\*args* și *\*\*kwargs*. Acesta este modul Pythonic de a specifica interpretorului că ne așteptăm să avem un număr variabil de parametri poziționali (*\*args*) și un număr variabil de parametri definiți folosind „nume = valoare”.

Din cate puteți vedea, decoratorii sunt o particularitate puternică a limbajului, care ne permite multă flexibilitate în implementarea funcțiilor permițând-ne să adăugăm multe funcționalități suplimentare în codul nostru într-un mod modular și reutilizabil.

### 3.1.5. Introducere în programarea orientată pe obiect folosind Python

În secțiunile anterioare am văzut cum putem folosi diferite structuri de date și obiecte deja existente în Python. În practică, există multe situații unde rezolvarea unei probleme devine mult mai simplă dacă folosim o structura de date care reprezintă mai eficient realitatea. De exemplu, imaginați-vă un program care determină cel mai scurt drum dintre doua orașe. Dacă folosim structurile de date pe care le-am întâlnit până acum ar trebui să avem o matrice de dimensiunea  $n \times n$ , unde  $n$  este numărul de orașe, în care să stocăm în fiecare celulă distanța

dintre cele doua orașe (în cazul în care acestea sunt conectate). Ulterior, trebuie să parcurgem această matrice de mai multe ori sărind de la o celulă la alta până când găsim distanța cea mai scurtă. Această problemă ar putea fi rezolvată mult mai ușor dacă am beneficia de o structură de date specială unde să stocăm o lista de orașe și pentru fiecare oraș să stocăm legăturile cu toate celelalte orașe. Din fericire, Python are funcționalitatea necesară pentru implementarea și folosirea unei structuri de date noi, specializate pentru problema pe care dorim să o rezolvăm, cu ajutorul claselor.

O clasă este un tipar pentru o anumită structură sau tip de date. Pe lângă atributele acelei structuri de date (datele ce pot fi stocate), o clasă definește și o serie de metode pentru a specifica cel puțin modul de inițializare a unui obiect și alte funcționalități sau transformări permise pentru datele conținute în acel obiect. Mai mult, printr-o clasă putem defini și tiparul interacțiunilor dintre obiecte de același tip sau de tipuri diferite. Cu alte cuvinte, în Python, o clasă este un șablon pentru crearea de obiecte. Acesta definește atributele și comportamentele care vor fi împărtășite de toate obiectele de acest tip.

Pentru a defini o clasă în Python, utilizăm cuvântul cheie *class* urmat de numele clasei. De obicei folosim convenția PascalCase când definim o clasă. Definiția clasei ar trebui să includă o metodă `__init__()`. Această metodă este specială în Python deoarece este apelată atunci când este creat un obiect de acel tip. În general, metoda `__init__()` este folosită pentru a inițializa atributele obiectului, dar poate fi folosită și pentru a configura anumite valori sau comportamente fundamentale ale obiectului. În exemplul de cod 15 vom defini o clasă `Vector3D` care are trei atribute, *x*, *y*, *z*. De asemenea, aceasta clasă implementează o metodă numită `lungime()` care calculează lungimea unui vector. La final, vom crea doi vectori și vom accesa atributele și metodele acestora.

```
# Clase si obiecte
# Definitia unei clase Vector3D pentru a reprezenta un vector in 3D
class Vector3D:

    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def lungime(self):
        return (self.x ** 2 + self.y ** 2 + self.z ** 2) ** 0.5

# Initializarea a doua obiecte de tip Vector3D
vector1 = Vector3D(1, 1, 1)
vector2 = Vector3D(1, 5, 10)

# Accesarea atributelor unui obiect Vector3D
vector1.x = 10
print(vector1.x)
>> 10

# Accesarea metodelor unui Vector3D
lungimea_vectorului1 = vector1.lungime()
```

```
lungimea_vectorului2 = vector2.lungime()
print(f"Lungimea vectorului 1 este {lungimea_vectorului1}.")
>> Lungimea vectorului 1 este 10.099504938362077.

print(f"Lungimea vectorului 2 este {lungimea_vectorului2}.")
>> Lungimea vectorului 2 este 11.224972160321824.
```

### Exemplul de cod 15. O clasă Vector3D.

Cuvântul cheie *self* este folosit pentru a se referi la instanța curentă a unei clase. Este folosit pentru a accesa atributele și metodele clasei din definițiile metodei. În exemplul de cod 15, *self* este folosit pentru a seta valorile *x*, *y* și *z* în metoda `__init__()` și pentru a le accesa în metoda `lungime()`.

Când am create o instanță a clasei Vector3D și am apelat o metodă a acesteia, Python va transmite automat instanța metodei ca prim argument precum în exemplul de mai sus. Este important să realizați că *self* nu este un cuvânt cheie rezervat în Python și nu este obligatoriu să îl utilizăm ca nume pentru primul argument dintr-o metodă. Cu toate acestea, este o convenție larg acceptată să folosim *self* ca nume pentru acest argument și este recomandat să urmăm această convenție în propriul cod.

Mai mult, în exemplul de mai sus ați observat o metodă cu un nume mai ciudat, `__init__()`. În Python aceste metode sunt cunoscute sub numele de metodele „dunder” (prescurtare de la „duble underscore”). Ele sunt metode speciale care au un nume și încep și se termină cu două caractere de subliniere. Rolul metodelor „dunder” este de a specifica interpretorului că dorim să implementăm un anumit comportament special, precum inițializarea, comparația sau adunarea. Aceasta i-a făcut pe unii programatori să le denumească și metode „magice” ce ne ajută să definim comportamentul anumitor operatori din clasele noastre.

De exemplu, metoda `__init__()` este o metodă dunder care este apelată atunci când este creat un obiect. Este folosit pentru a inițializa atributele obiectului. Metoda `__str__()` este o altă metodă dunder care este folosită pentru a defini modul în care un obiect ar trebui să fie reprezentat ca text. În exemplul de cod 16 vom extinde funcționalitatea clasei *Vector3D* cu ajutorul metodelor magice.

```
# Definitia unei clase Vector3D pentru a reprezenta un vector in 3D
class Vector3D:
```

```
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def lungime(self):
        return (self.x ** 2 + self.y ** 2 + self.z ** 2) ** 0.5

    def __str__(self):
        return f"x = {self.x}. y = {self.y}. z = {self.z}."
```

```

def __add__(self, vector):
    x = self.x + vector.x
    y = self.y + vector.y
    z = self.z + vector.z
    rezultat = Vector3D(x, y, z)
    return rezultat

def __sub__(self, vector):
    x = self.x - vector.x
    y = self.y - vector.y
    z = self.z - vector.z
    rezultat = Vector3D(x, y, z)
    return rezultat

def __mul__(self, vector):
    return self.x * vector.x + self.y * vector.y + self.z * vector.z

def __eq__(self, vector):
    return self.x == vector.x and self.y == vector.y and self.z == vector.z

def __neg__(self):
    return Vector3D(-self.x, -self.y, -self.z)

def __ne__(self, vector):
    return self.x != vector.x or self.y != vector.y or self.z != vector.z

# Initializarea a doua obiecte de tip Vector3D
vector1 = Vector3D(1, 1, 1)
vector2 = Vector3D(1, 5, 10)

# Adunarea a doi vector
suma = vector1 + vector2
print(suma)
>> x = 2. y = 6. z = 11.

# Diferenta a doi vectori
diferenta = vector1 - vector2
print(diferenta)
>> x = 0. y = -4. z = -9.

# Produsul scalar
scalar = vector1 * vector2
print(scalar)
>> 17

```

Exemplul de cod 16. Extensia clasei Vector3D folosind metode “magice”.

Metodele „dunder” nu sunt singura funcționalitate puternică a claselor în Python. Pe lângă metode magice, există și atribute și metode ale clasei. Atributele clasei sunt variabile care sunt asociate cu o clasă și sunt „împărțite” de toate instanțele acelei clase. Ele sunt definite la nivel de clasă, în afara oricărei definiții de metodă și sunt de obicei accesate la fel ca un atribut al unui obiect. În general folosim atributele clasei pentru stocarea informațiilor care sunt partajate de toate instanțele unei clase, cum ar fi setările de configurare sau constantele. Ele pot fi utile

și pentru setarea valorilor implicite. Cu toate acestea, este recomandat să folosim atribute de instanță ori de câte ori este posibil, mai degrabă decât să ne bazăm pe atributele clasei.

O metodă de clasă este o metodă care este asociată cu o clasă și este accesată folosind decoratorul `@classmethod`. Metodele de clasă sunt definite la nivel de clasă, în afara oricărei definiții de metodă și sunt accesate ca și metodele unei instanțe. În exemplul de cod 17 putem vedea o metodă de clasă `get_dimensions()` partajată de toate obiectele `Vector3D`. Aceasta este definită la nivel de clasă și este accesată folosind notația cu puncte pe obiecte. Decoratorul `@classmethod` indică faptul că aceasta este o metodă de clasă. În plus, primul argument al unei metode de clasă este `cls`, care se referă la clasa în sine. Acest lucru permite metodei clasei să acceseze atributele clasei și alte informații la nivel de clasă.

Metodele de clasă sunt utile pentru a crea metode care sunt asociate cu o clasă, mai degrabă decât cu o instanță specifică a clasei. Ele pot fi utilizate pentru o varietate de scopuri, cum ar fi crearea de metode din fabrică (metode care creează instanțe noi ale unei clase) sau modificarea stării la nivel de clasă. În general, metodele de clasă sunt un instrument util pentru crearea de cod reutilizabil și modular în Python.

În afara metodelor și atributelor de clasă, Python ne oferă și metode statice. Aceste metode sunt asociate cu o clasă și sunt definite folosind decoratorul `@staticmethod`. Metodele statice sunt definite la nivel de clasă, în afara oricărei definiții de metodă și sunt de obicei accesate folosind notația cu puncte, la fel ca în cazul obiectelor. Spre deosebire de metodele de clasă sau de instanță, metodele statice nu primesc niciun prim argument special. Aceste metode sunt pur și simplu funcții care sunt asociate cu o clasă, dar nu au nicio conexiune specifică cu clasa sau cu instanțele acesteia.

Metodele statice sunt utile pentru a crea funcții utilitare care sunt asociate cu o clasă, dar nu trebuie să acceseze nicio stare specifică clasei. Ele pot fi folosite pentru a încapsula cod care îndeplinește o anumită sarcină și pot fi reutilizate în contexte diferite.

```
class Vector3D:
    dimensions = 3

    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def lungime(self):
        return (self.x ** 2 + self.y ** 2 + self.z ** 2) ** 0.5

    @classmethod
    def get_dimensions(cls):
        return cls.dimensions

    @classmethod
    def from_list(cls, lista):
        return Vector3D(lista[0], lista[1], lista[2])
```

```
# Initializarea a doua obiecte de tip Vector3D
vector1 = Vector3D(1, 1, 1)
vector2 = Vector3D(1, 5, 10)

# Print dimensions
print(f"Vector 1 dimensions = {vector1.dimensions}")
>> Vector 1 dimensions = 3

print(f"Vector 2 dimensions = {vector2.dimensions}")
>> Vector 2 dimensions = 3

print(f"Vector3D dimensions = {Vector3D.get_dimensions()}")
>> Vector3D dimensions = 3

# Initializeaza un obiect Vector3D dintr-o lista
vector3 = Vector3D.from_list([1, 1, 2])
print(vector3)
>> x = 1. y = 1. z = 2.
```

Exemplul de cod 17. Extensia clasei Vector3D folosind attribute și metode de clasă.

În ultima parte a acestei secțiuni vom discuta despre moștenire. În Python, moștenirea este o modalitate de a crea o nouă clasă care este o versiune modificată a unei clase existente. Noua clasă se numește subclasa, iar clasa existentă este superclasa.

Moștenirea permite subclasei să moștenească attribute și metode din superclasă, ceea ce poate economisi timp și efort atunci când ne proiectăm clasele. De asemenea, ne permite să reutilizăm codul și să realizăm o bază de cod mai organizată și mai ușor de întreținut. În general, moștenirea este o caracteristică puternică a programării orientate pe obiecte. Dacă doriți să aflați mai multe despre programarea orientată pe obiect, puteți consulta (Ramalho, 2015).

Pentru a crea o subclasă în Python, utilizăm cuvântul cheie *class* urmat de numele subclasei și apoi specificăm numele superclasei în paranteze. În exemplul de cod 18 vom extinde clasa Vector3D la patru dimensiuni. Noua clasă, Vector4D moștenește metodele `__init__()` și `get_dimensions()` din clasa Vector3D, dar are și propria sa metodă `__init__()` și o versiune modificată a metodei `lungime()`.

```
class Vector3D:

    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
        self.dimensions = 3

    def lungime(self):
        return (self.x ** 2 + self.y ** 2 + self.z ** 2) ** 0.5

    def get_dimensions(self):
        return self.dimensions
```



```

class Vector4D(Vector3D):

    def __init__(self, x, y, z, t):
        super(Vector4D, self).__init__(x, y, z)
        self.t = t
        self.dimensions = 4

    def lungime(self):
        return (self.x ** 2 + self.y ** 2 + self.z ** 2 + self.t ** 2) ** 0.5

# Initializarea a doua obiecte de tip Vector3D
vector1 = Vector4D(1, 1, 1, 2)
vector2 = Vector4D(1, 5, 10, 25)

# Accesarea metodelor unui Vector3D
lungimea_vectorului1 = vector1.lungime()
lungimea_vectorului2 = vector2.lungime()
print(f"Lungimea vectorului 1 este {lungimea_vectorului1}.")
>> Lungimea vectorului 1 este 2.6457513110645907.

print(f"Lungimea vectorului 2 este {lungimea_vectorului2}.")
>> Lungimea vectorului 2 este 27.40437921208944.

# Dimesiunea vectorului Vector4D
print(vector1.get_dimensions())
>> 4

```

Exemplul de cod 18. Exemplu de moștenire în Python.

### 3.2. Python pentru învățarea automată

Python este un limbaj popular pentru învățarea automată, deoarece oferă o serie de biblioteci și cadre utile care facilitează implementarea algoritmilor și tehnicilor de învățare automată.

Unele dintre caracteristicile cheie ale limbajului Python care îl fac bine potrivit pentru învățarea automată sunt:

- O comunitate mare și activă de dezvoltatori: Python are o comunitate mare și activă de dezvoltatori, ceea ce înseamnă că există multe biblioteci și cadre disponibile pentru învățarea automată, precum și o mulțime de resurse și suport online.
- O sintaxă expresivă și lizibilă: Python are o sintaxă simplă și expresivă, ceea ce face ușor de citit și de scris cod. Acest lucru este util în special pentru învățarea automată, unde accentul se pune adesea pe înțelegerea și interpretarea datelor și a rezultatelor, mai degrabă decât pe mecanica implementării.
- O gamă largă de biblioteci și pachete: Python are o serie de biblioteci și cadre care sunt special concepute pentru învățarea automată, cum ar fi NumPy, Pandas și scikit-learn.

Aceste biblioteci oferă o serie de instrumente și funcții care facilitează implementarea algoritmilor și tehnicilor de învățare automată în Python.

- Performanță bună: în ciuda simplității și lizibilității sale, Python este, de asemenea, un limbaj de înaltă performanță și este capabil să gestioneze seturi mari de date și modele complexe de învățare automată.
- Suport pentru calculul distribuit: Python are, de asemenea, suport încorporat pentru calculul distribuit, ceea ce facilitează scalarea algoritmilor și modelelor de învățare automată pentru a profita de resursele de calcul puternice

În general, combinația Python dintre o comunitate mare și activă, sintaxa expresivă și un set bogat de biblioteci și cadre îl fac o alegere populară pentru sarcinile de învățare automată. În restul secțiunii vom explora cele mai întâlnite biblioteci folosite pentru a dezvolta soluții de învățare automată.

### 3.2.1. Numpy

NumPy<sup>19</sup> este o bibliotecă Python pentru lucrul cu matrice mari, multidimensionale și matrice de date numerice. Este un instrument puternic pentru calculul științific și matematic, care oferă operații rapide și eficiente pe matrice.

NumPy este deosebit de util pentru efectuarea de operații matematice pe seturi mari de date, deoarece ne permite să efectuăm operații pe matrice și matrice întregi, mai degrabă decât să trebuiască să parcurgem manual fiecare element al matricei. Acest lucru poate fi mult mai rapid și mai eficient decât utilizarea tipurilor de secvențe încorporate din Python, cum ar fi listele și tuplele.

NumPy oferă, de asemenea, un număr mare de funcții pentru efectuarea de operații matematice comune, cum ar fi algebra liniară, analiza statistică și generarea de numere aleatorii. De asemenea, include instrumente pentru integrarea cu alte biblioteci științifice, cum ar fi SciPy și Matplotlib.

În exemplul de cod 19 vedem cum putem folosi NumPy pentru a efectua diferite operații matematice cu matrice. Întregul exemplu începe prin a crea două matrice 5 x 5 de numere aleatorii între 0 și 1. Ulterior vom calcula suma și produsul celor două matrice.

```
# Exemplu NumPy
import numpy as np
```

```
# Initializarea a doua matrice 5 x 5
A = np.random.random((5, 5))
```

---

<sup>19</sup> <https://numpy.org>

```

B = np.random.random((5, 5))

# Inmultire
C = np.matmul(A, B)
print(C)
>> [[0.81743149 1.74079954 0.64831392 0.97799284 1.74147594]
      [0.72598281 1.63124079 0.64796992 1.08009896 1.33676989]
      [0.84028543 1.96733834 0.72069461 1.1316824 1.77445639]
      [0.6580626 1.55866818 0.52908098 0.9337196 1.27642513]
      [1.02412878 2.37051918 1.00961651 1.55058674 1.92581327]]

# Adunare
D = A + B
print(D)
>> [[0.53246968 1.18053966 1.26360696 1.27593746 0.90106446]
      [1.03778211 0.93217646 1.01158536 1.51777923 0.32348069]
      [0.68807001 1.1482194 0.92194362 0.65676714 1.28925539]
      [0.65136548 0.51596716 1.04055539 0.16087844 1.54351967]
      [1.27191792 1.49480566 1.4176902 1.61254889 0.87544121]]

```

Exemplul de cod 19. Manipularea matricelor folosind NumPy.

În general, NumPy este un instrument valoros pentru calculul științific și matematic în Python, deoarece oferă operații rapide și eficiente pe matrice mari de date.

### 3.2.2. Scipy

SciPy<sup>20</sup> este o bibliotecă Python pentru calcul științific care oferă o gamă largă de algoritmi și funcții pentru lucrul cu date științifice. Este construit pe NumPy și oferă multe dintre aceleași capacități, dar include și funcții suplimentare pentru sarcini precum optimizarea, interpolarea și procesarea semnalului și a imaginii.

SciPy este deosebit de util pentru sarcini care implică calcule numerice complexe, cum ar fi optimizarea, interpolarea și analiza statistică. Acesta oferă un număr mare de funcții și algoritmi care sunt proiectați să fie eficienți și fiabili pentru aceste sarcini.

În exemplul de cod 20 vedem cum putem folosi SciPy pentru a efectua o descompunere LU a unei matrice de numere aleatorii. Întregul exemplu începe prin a crea o matrice 3 x 3 de numere aleatorii între 0 și 1. Ulterior vom folosi SciPy pentru a descompune matricea în două matrice L, U.

```

# Exemplu SciPy
from scipy import linalg
import numpy as np

# Initializarea unei matrice 3 x 3

```

---

<sup>20</sup> <https://scipy.org>

```

A = np.random.random((3, 3))

# LU Decomposition
P, L, U = linalg.lu(A)
print(P)
>> [[0. 0. 1.]
      [0. 1. 0.]
      [1. 0. 0.]]

print(L)
>> [[1.    0.    0.    ]
      [0.97717362 1.    0.    ]
      [0.95428002 0.43525627 1.    ]]

print(U)
>> [[ 0.64404409  0.05497356  0.67471749]
      [ 0.    0.5826711  -0.04172695]
      [ 0.    0.    0.21336312]]

```

Exemplul de cod 20. Factorizarea matricelor folosind SciPy.

În concluzie, SciPy este o unealta foarte eficientă pentru calculul științific ce oferă o paletă foarte largă de algoritmi pentru sarcini precum optimizarea, interpolarea și procesarea semnalelor și a imaginilor.

### 3.2.3. Pandas

Pandas<sup>21</sup> este o bibliotecă Python pentru manipularea și analiza datelor. Oferă o gamă largă de instrumente și funcții pentru lucrul cu date tabelare, cum ar fi datele stocate într-o foaie de calcul sau o bază de date.

Pandas este util pentru sarcini precum curățarea și pregătirea datelor, manipularea datelor și efectuarea de analize statistice. Oferă o serie de funcții și structuri de date care facilitează manipularea și analiza datelor în Python.

Una dintre caracteristicile cheie ale Pandas este obiectul DataFrame, care este un tabel bidimensional de date cu rânduri și coloane. DataFrame-urile sunt similare cu foile de calcul Excel și pot fi folosite pentru a stoca și manipula date tabelare. Acestea oferă o serie de funcții pentru selectarea și filtrarea datelor, precum și pentru efectuarea de operațiuni asupra datelor, cum ar fi sortarea, gruparea și agregarea.

În exemplul de cod 21 vedem cum putem să citim datele dintr-un fișier „.csv” și ulterior să obținem o descriere rapidă a datelor conținute în acest fișier. Ulterior selectăm un subset de

---

<sup>21</sup> <https://pandas.pydata.org>

date în baza unor condiții și le grupăm după o variabilă categorică pentru a calcula media fiecărui grup.

```
# Pandas example
import pandas as pd

# Citeste datele dintr-un fisier .csv
data = pd.read_csv("weather.csv")

# Descrie datele numerice din fisier
description = data.describe()

# Selecteaza randurile pentru care temperatura este mai mare de 5 grade C
subset = data[data["Temperature (C)"] > 5.0]

# Grupeaza datele in functie de tipul precipitatiei si calculeaza media pentru fiecare
mean_temperature = data.groupby("Precip Type")["Temperature (C)"].mean()
print(f"Temperatura medie este: ")
print(mean_temperature)
>> Temperatura medie este:
Precip Type
rain    13.852989
snow    -3.270885
Name: Temperature (C), dtype: float64
```

Exemplul de cod 21. Manipularea datelor tabelare cu Pandas.

În general, Pandas este o bibliotecă puternică și utilizată pe scară largă pentru manipularea și analiza datelor în Python și este un instrument valoros pentru sarcini de analiza descriptivă și statistică a datelor tabelare. În plus, Pandas oferă și câteva opțiuni de vizualizare a datelor și poate fi integrată cu diferite baze de date și tipuri de fișiere facilitând astfel accesul la date.

### 3.2.4. Scikit-learn

Scikit-learn<sup>22</sup> este o bibliotecă Python pentru învățarea automată care oferă o gamă largă de algoritmi și funcții pentru sarcini precum clasificarea, regresia, gruparea și reducerea dimensionalității. Este construit pe NumPy și SciPy și este conceput pentru a fi ușor de utilizat și eficient pentru aceste sarcini.

Scikit-learn este deosebit de util pentru sarcini care implică antrenarea unui model de învățare automată pe un set de date și realizarea de predicții pe baza modelului respectiv. Oferă o interfață consistentă pentru o gamă largă de algoritmi de învățare automată și facilitează comutarea între diferite modele.

---

<sup>22</sup> <https://scikit-learn.org/stable/>

În exemplul de cod 22, încărcăm date despre iriși și le folosim pentru a antrena un model de regresie logistică folosind clasa *LogisticRegression* a scikit-learn. Apoi folosim modelul pentru a valida performanța acestuia pe setul de antrenare și pe cel de validare. În acest exemplu vom folosi acuratețea clasificatorului pentru a măsura cât de bun este clasificatorul nostru.

```
# scikit-learn example
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Incarcam datele
X, y = load_iris(return_X_y=True)

# Impartim datele intr-un set de antrenare si unul de validare
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=42)

# Antrenam un clasificator
clf = LogisticRegression()
clf = clf.fit(X_train, y_train)

# Validam performanta clasificatorului pentru setul de antrenare
y_train_predict = clf.predict(X_train)
train_accuracy = accuracy_score(y_true=y_train, y_pred=y_train_predict)
print(f"Accuratetea pe setul de antrenare este: {train_accuracy * 100}%.")
>> Accuratetea pe setul de antrenare este: 97.6923076923077%.

# Validam performanta clasificatorului pentru setul de validare
y_test_predict = clf.predict(X_test)
test_accuracy = accuracy_score(y_true=y_test, y_pred=y_test_predict)
print(f"Accuratetea pe setul de antrenare este: {test_accuracy * 100}%.")
>> Accuratetea pe setul de antrenare este: 100.0%.
```

Exemplul de cod 22. Clasificare folosind regresia logistică din scikit-learn.

În general, scikit-learn este o bibliotecă puternică și utilizată pe scară largă pentru învățarea automată în Python și este un instrument valoros pentru sarcini precum clasificarea, regresia, gruparea și reducerea dimensionalității.

### 3.2.5. Matplotlib

Matplotlib<sup>23</sup> este o bibliotecă de vizualizare a datelor implementată în Python. Permite utilizatorilor să creeze o gamă largă de vizualizări statice, animate și interactive în Python. Unul dintre motivele principale pentru utilizarea Matplotlib este că este foarte ușor de utilizat și oferă o interfață de nivel înalt pentru crearea unei varietăți de grafice. De asemenea, este foarte personalizabil, permițându-ne să specificăm culorile, stilurile de linii și alte opțiuni de formatare pentru graficele noastre. În plus, Matplotlib poate fi folosit pentru a crea o gamă

---

<sup>23</sup> <https://matplotlib.org>

largă de diagrame, inclusiv diagrame de linii, diagrame de dispersie, diagrame cu bare, bare de eroare, diagrame circulare și multe altele.

Matplotlib este util pentru analiza și vizualizarea datelor într-o gamă largă de domenii, inclusiv finanțe, biologie, științe sociale și inginerie. De asemenea, este utilizat în mod obișnuit în calculul științific și știința datelor, unde este adesea folosit pentru a vizualiza rezultatele simulărilor sau analizelor.

În exemplul de cod 23 vom crea un grafic al unei funcții definită pe intervalul  $[-10, 10]$ . Pentru acest exemplu vom folosi NumPy cât și Matplotlib, cele două librării fiind adesea folosite împreună. Rezultatele codului pot fi vizualizate în Figura 8.

```
# Exemplu Matplotlib
import numpy as np
import matplotlib.pyplot as plt

# Definim o functie al carei grafic vrem sa il realizam
def traectorie(t):
    return t**2 + 2.5 * np.sin(3*t) + 10

# Definim intervalul pe care este reprezentata functia
t = np.linspace(-10, 10, 100)

# Calculam valoarea functiei in fiecare punct din intervalul specificat anterior
y = traectorie(t)

# Cream graficul functiei
_ = plt.figure(figsize=(10, 10))
plt.plot(t, y, marker="+", c="b", linestyle="--")
plt.legend(["Traectoria"])
plt.title("Graficul functiei traectorie()", fontsize=14)
plt.ylabel("t", fontsize=11)
plt.xlabel("y", fontsize=11)
plt.show()
```

Exemplul de cod 23. Realizarea graficului unei funcții folosind NumPy și Matplotlib.

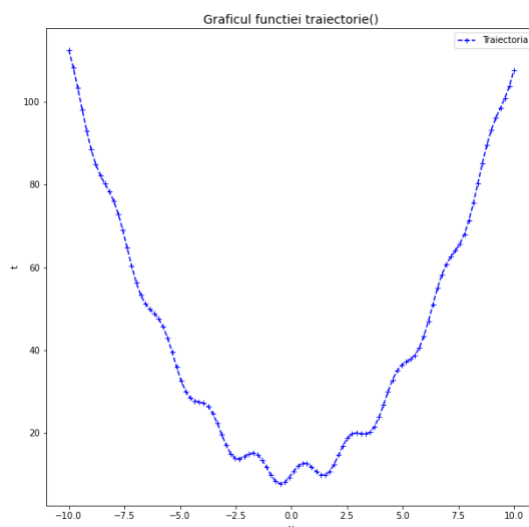


Figura 8. Graficul unei funcții realizat cu ajutorul NumPy și Matplotlib.

În general, Matplotlib este un instrument puternic și flexibil, care este utilizat pe scară largă pentru vizualizarea datelor în Python.

Din păcate, scopul acestei lucrări nu este să introducă toate detaliile acestor biblioteci. Dar, dacă sunteți interesați să aflați mai multe, puteți să consultați referințele prezentate în bibliografia acestui capitol sau puteți să încercați să rezolvați din exercițiile incluse.

Împreună, toate bibliotecile prezentate în această secțiune stau la baza multor soluții de calcul științific sau de învățare automată. Aceste unelte, dacă sunt folosite eficient, pot genera soluții foarte puternice.

## Exerciții

1. Implementați o funcție care determina dacă un număr întreg este par.
2. Implementați o funcție care determina dacă un număr natural este prim.
3. Implementați o funcție care determină dacă un sir de caractere este un palindrom.
4. Implementați o clasă *Figura* care simbolizează o figura geometrică regulată. Aceasta clasă va avea următoarele proprietăți și metode:
  - a. *nume* = un sir de caractere
  - b. *nr\_laturi* = un număr întreg specificând numărul de laturi ale figurii
  - c. *lungimea\_laturilor* = o listă ce conține lungimea tuturor laturilor figurii
  - d. *perimetru* = o metoda care calculează perimetrul figurii



5. Extindeți clasa *Figura* cu ajutorul moștenirii și realizați o clasă *Patrat*. În afara de metodele moștenite, clasa *Patrat* are o metodă proprie, *arie()*, care calculează aria pătratului.
6. Implementați eliminarea Gauss folosind NumPy, fără a folosi funcțiile existente în această bibliotecă.
7. Implementați metoda Newton pentru aflarea rădăcinii unei ecuații de forma  $f(x) = 0$ .
8. Realizați graficul funcției  $f(x) = 2x + 3$ ,  $-10 < x < 10$ , folosind NumPy și Matplotlib.
9. Realizați graficul funcției  $f(x) = \sqrt{e^x + x^3/10 - \sin 3x + \tan^{-1} 4x}$ ,  $-5 < x < 5$ , folosind NumPy și Matplotlib.

## Bibliografie

- Abelson, H., Sussman, G. J., & Sussman, J. (1996). *Structure and interpretation of computer programs*. Cambridge, Massachusetts: MIT Press. ISBN 9780262510875.
- Laakmann, G. (2020). *Cracking the coding interview*. Palo Alto: CareerCup, LLC.
- Ramalho, L. (2015). *Fluent Python: Clear, Concise, and Effective Programming*. Sebastopol, CA: O'Reilly Media Inc. ISBN 9781492056355.
- McKinney, W. (2018). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. Sebastopol, CA: O'Reilly Media Inc. ISBN 978491957660.
- Vanderplass J. T. (2016). *Python Data Science Handbook: Essential Tools for Working with Data*. Sebastopol, CA: O'Reilly Media Inc. ISBN 1491912057.
- Johannson, R. (2019). *Numerical Python: Scientific Computing and Data Science Applications with NumPy, SciPy and Matplotlib*. Apress Media LLC. ISBN 9781484242452.

## 4. Învățarea supravegheată

### 4.1. Noțiuni fundamentale

În învățarea supravegheată, probabil cea mai des întâlnită formă de învățare automată, ne confruntăm cu următoarea problemă.

Avem un sistem<sup>24</sup> și un set de date corespunzând unor observații despre componentele sistemului în anumite situații<sup>25</sup>. Aceste observații constituie un set de exemple etichetate. Fiecare observație este la rândul ei alcătuită din mai multe componente, fiecare componentă

<sup>24</sup> În acest suport de curs, un sistem poate fi un fenomen fizic, social, cognitiv, un proces natural sau de business, sau orice alta situație pe care dorim să o modelăm matematic.

<sup>25</sup> În acest context, o situație se poate referi la un moment de timp sau o anumită locație în spațiu.

corespunzând unei variabile pe care o considerăm relevantă pentru a descrie legătura dintre aceste observații și eticheta aferentă.

De exemplu, în cazul recunoașterii obiectelor din figura 1, am putea avea pentru fiecare imagine o etichetă cu obiectele prezente în acea imagine. Mai mult, în cazul resortului din figura 2, am putea eticheta fiecare măsurătoare a forței, masei, momentului când efectuăm o măsurătoare, și a constantei elastice cu lungimea resortului. Aceste observații, luate împreună, constituie un eșantion<sup>26</sup>.

În funcție de tipul etichetelor pe care le atribuim unei observații<sup>27</sup>, putem distinge două situații (precum observăm și din exemplele din figurile 1 și 2). Dacă etichetele formează o mulțime cu un număr finit de posibile valori<sup>28</sup> avem de a face cu o clasificare. În caz contrar, când etichetele formează o mulțime cu un număr infinit de valori sau orice valoare dintr-un interval prestabilit, avem de rezolvat o problemă de regresie.

Să recapitulăm unde am ajuns. Avem un set de observații pe care de acum înainte îl vom numi **X**. Aceste observații reprezintă o colecție pe care o putem reprezenta matematic sub forma unei matrice precum cea din ecuația [E1]. În exemplul de mai jos avem o matrice cu  $m$  observații, fiecare observație fiind alcătuită din  $n$  componente.

$$\begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix} \text{ [E1]}$$

O singură observație poate fi reprezentată sub forma unui vector<sup>29,30</sup>, precum cel din ecuația [E2].

$$\vec{x}_i = (x_{i1}, x_{i2}, \dots, x_{in}) \text{ [E2]}$$

Este important să ne gândim puțin ce reprezintă aceste date. Ele sunt niște măsurători efectuate asupra unor părți componente ale sistemului pe care le presupunem ca fiind reprezentative pentru a descrie toate comportamentele de interes (uneori chiar posibile) ale acestuia. Un alt mod de a privi aceste date este să ne gândim la ele reprezentă un instantaneu, o „fotografie” pe care o facem sistemului într-un anumit moment. Din punct de vedere matematic, aceste observații alcătuiesc un spațiu vectorial format din  $n$  dimensiuni.

---

<sup>26</sup> În funcție de persoana cu care discutați și de formarea academică a acestuia, puteți găsi diferite denumiri pentru colecția de observații. În acest suport de curs vom folosi denumirea de eșantion.

<sup>27</sup> A se înțelege ca o observație este alcătuită din una sau mai multe cantități sau caracteristici măsurate.

<sup>28</sup> Mulțimea valorilor etichetelor poate fi foarte mare. Este important de observat că pentru ca o problemă să fie catalogată drept o clasificare este necesar ca aceste valori să fie numărabile (chiar dacă sunt foarte multe)

<sup>29</sup> Pentru moment gândiți-vă la un vector ca la o colecție de elemente.

<sup>30</sup> În acest suport de curs veți găsi vectorii reprezentați și folosind litere mici, în italic și bold,  $\mathbf{x}_i$

Pe lângă această mulțime de măsurători, avem și etichetele corespunzătoare fiecărei observații. În cele ce urmează ne vom referi la aceste etichete drept  $y$ . Această mulțime are o corespondență unu la unu cu observațiile  $X$ . Prin urmare, putem concluziona că avem următoarele informații:

- Un set de date,  $X$ , corespunzând observațiilor făcute
- Un set de date,  $y$ , corespunzând etichetelor fiecărei observații din  $X$

Putem reprezenta aceste seturi de date ca un întreg, ca o mulțime formată din dubletele  $(\vec{x}_i, y_i)$ . În limbaj matematic, această idee se poate transcrie folosind notația din ecuația [E3].

$$\{(\vec{x}_i, y_i)\}, i = \{1, 2, 3, \dots, m\} \text{ [E3]}$$

Această reprezentare este destul de opacă și nu pare să ne ofere o modalitate intuitivă prin care să înțelegem ceea ce facem. În cele ce urmează vom încerca să disecăm puțin ce ne spun aceste ecuații din punct de vedere geometric, plecând de la următoarea situație.

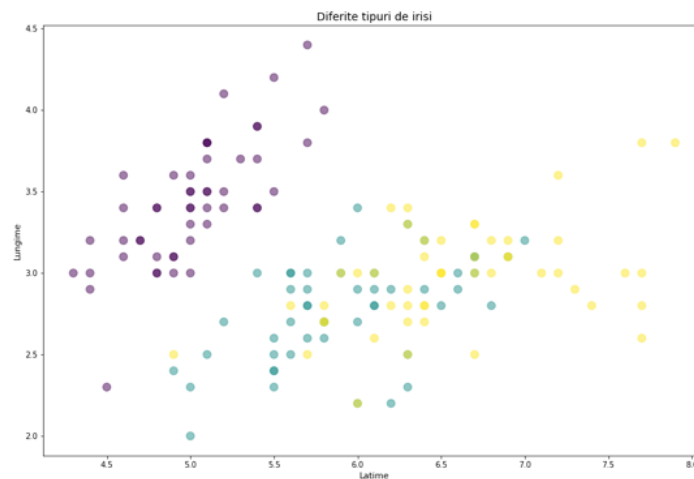


Figura 9. O reprezentare grafică a observațiilor legate de lungimea și lățimea sepalei a unui grup de iriși.

Dacă doriți să realizați același grafic, puteți folosi codul din exemplul de cod 24 unde am folosit *scikit-learn* și *matplotlib*.

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X = X[:, :2]

_ = plt.figure(figsize=(15, 10))
plt.scatter(X[:, 0], X[:, 1], c=y, s=100, alpha=0.5)
plt.title("Diferite tipuri de irisi", fontsize=14)
plt.ylabel("Lungime", fontsize=11)
plt.xlabel("Latime", fontsize=11)
plt.show()
```

## Exemplul de cod 25. Generarea diagramei de dispersie pentru a vizualiza variația dintre lungimea și lățimea sepalei a trei tipuri de iriși.

Avem o mulțime de iriși (da, ați citit bine, iriși, florile). Pentru fiecare iris din colecția noastră măsurăm lungimea și lățimea sepalei și etichetăm fiecare observație cu tipul de iris corespunzător<sup>31</sup>. Nimic nou pana acum, doar am construit setul de date descris de ecuația [E3].

Dorind să înțelegem mai bine ce reprezintă acele numere, le vom reprezenta într-o formă grafică ca în figura 9. După cum putem observa, fiecare observație este reprezentată de un punct într-un spațiu 2D format de valorile posibile pentru lungimea și respectiv lățimea sepalei unui iris. În practică, de cele mai multe ori, nu avem acest lux și trebuie să operăm într-o lume cu mult mai multe dimensiuni. Mai mult, tot în figura 9 putem observa că în funcție de tipul de iris, fiecare punct primește o anumită culoare (o etichetă). Prin etichetarea observațiilor am creat niște „insule” în acest spațiu 2D unde fiecare insulă corespunde unui anumit tip de iris. De exemplu, punctele mov din partea stângă a graficului ne oferă o indicație asupra posibilelor dimensiuni pe care le pot avea petalele irișilor de tipul 1. Mai mult, dacă cineva ne-ar oferi acum informații despre un iris necunoscut, am putea intuitiv să îl încadrăm în una din cele trei specii cunoscute doar bazat pe poziția punctului în acest spațiu 2D creat din lungimea și lățimea sepalei.

Revenind la problema inițială, putem concluziona că setul de date alcătuit din observații și etichete ne oferă o privire asupra potențialelor conexiuni dintre acestea. Cu alte cuvinte, plecând de la acest set de date putem să descoperim regulile de funcționare ale sistemului prin descoperirea mapării dintre observații și etichete.

Aceasta mapare constituie primul pas spre a realiza o mașină capabilă să înțeleagă diferite fenomene, procese, sau situații. Dar, avem totuși o problemă pe care nu am încercat să o rezolvăm încă, și anume, cum arată această mapare (daca într-adevăr ea există<sup>32</sup>)? După cum ne-am obișnuit, nici o întrebare nu vine singură, iar situația de față nu este o excepție.

Presupunând că există o mapare între observații și etichete, aceasta nu ne garantează că toate componentele observațiilor sunt relevante. Este posibil ca doar o submulțime a acestora să fie reprezentative, celelalte provocând mai mult rău decât bine. De asemenea, avem situații în care nu am reușit să includem toate componentele necesare. În aceste cazuri, ce componente ar trebui să mai adăugăm? Toate aceste situații și întrebări se vor regăsi în majoritatea problemelor

---

<sup>31</sup> Acest set de date este deja consacrat în învățarea supravegheată. Pentru mai multe detalii despre setul de date și cum a fost generat, navigați aici: <https://archive.ics.uci.edu/ml/datasets/iris>

<sup>32</sup> În acest suport de curs nu ne vom pune întrebări în legătură cu existența acestei mapări. Vom considera că ea există și că poate fi descoperită folosind datele disponibile. În realitate însă, putem avea situații în care observațiile să fie irelevante și să nu putem descoperi o mapare reală între acestea și etichetele atribuite. Mai mult, există situații în care această mapare este foarte complicată sau observațiile conțin erori foarte mari. Atunci avem nevoie de un număr foarte mare de observații și/sau de metode mai avansate decât cele prezentate în acest capitol pentru a putea descoperi regulile de funcționare ale sistemului.

practice pe care încercăm să le rezolvăm cu ajutorul învățării supravegheate<sup>33</sup> (și a învățării automate în general). Însă, în cele ce urmează voi face ceea ce fac toți autorii când vor să evite întrebări dificile; mă voi prefăca că aceste întrebări nu există și voi presupune că maparea există, că toate variabilele dintr-o observație sunt relevante și că măsoară suficiente variabile pentru a putea găsi acea mapare.

Cum găsim aceasta mapare? Începem aceasta aventură în contextul unei probleme de regresie. Avem o mulțime de  $m$  observații  $\{(\vec{x}_i, y_i)\}, i = \{1, 2, 3, \dots, m\}$  unde  $\mathbf{x}$  are  $n$  componente și  $y$  poate lua orice valoare din intervalul  $(y_{min}, y_{max})$ .

În realitate nu putem descoperi maparea reală dintre observații și etichete. Aceasta rămâne ascunsă și necunoscută nouă deoarece nu avem acces decât la informații parțiale despre sistem (un număr limitat de observații, chiar dacă acest număr este foarte mare). Prin urmare, trebuie să ne mulțumim cu cea mai bună alternativă a acesteia. Dar care ar putea fi aceasta și cum am putea proceda să o găsim?

Pentru a răspunde la aceste întrebări, haideți să încercăm să formalizăm puțin problema în speranța că vom putea folosi metode matematice pentru a o descoperi.

Ce avem până acum?

1. Avem un set de date, de exemplu,  $\{(\vec{x}_i, y_i)\}, i = \{1, 2, 3, \dots, m\}$
2. Știm, sau mai bine spus, presupunem că există o mapare dintre observații,  $\mathbf{x}$ , și etichete,  $y$ . Vom nota această mapare cu  $h(\vec{x})$ , pentru orice valoare  $\mathbf{x}$
3. Maparea reală ne este necunoscută, dar știm că pentru anumite valori ale lui  $\mathbf{x}_i$ , aceasta are valoarea  $y_i$  (avem această informație din setul nostru de date etichetate)

Ce vrem să realizăm? Vrem să găsim cea mai bună alternativă pentru  $h(\vec{x})$ , pe care o vom numi  $\hat{h}(\vec{x})$ .

Plecând de la aceasta premisă, haideți să stabilim întâi ce înțelegem prin cea mai bună alternativă. Ne va fi destul de greu să determinăm această mapare dacă nu definim de pe acum ce înseamnă cea mai bună alternativă. Intuitiv, am putea considera că  $\hat{h}(\vec{x})$  este maparea care reușește să facă cât mai puține erori atunci când transformă o observație cunoscută într-o etichetă. Cu alte cuvinte,  $\hat{h}(\vec{x})$  este maparea care reușește să transforme observațiile din setul nostru de date în etichetele corespunzătoare făcând cât mai puține erori în acest proces.

---

<sup>33</sup> Observați că nu vom putea niciodată să avem un răspuns absolut la aceste întrebări deoarece nu avem acces la toate comportamentele posibile ale sistemului. Întreaga metodologie se bazează pe un eșantion de observații (care de cele mai multe ori conțin erori sau sunt parțial adevărate), prin urmare nu avem cum să găsim cu certitudine răspunsul la aceste întrebări.

Folosind acest raționament, putem specifica acum ce înseamnă că  $\hat{h}(\vec{x})$  face o greșeală. Avem nevoie de această definiție pentru a putea contoriza cât de bine reușește să mapeze observațiile cunoscute.

Și în această situație ne putem baza pe intuiție sau putem căuta rapid în literatura de specialitate. Însă, pentru scopul acestei lucrări, mă voi rezuma la una dintre cele mai simple metode, și anume, vom cuantifica cât de greșită este o mapare,  $\hat{y}$ , față de valoarea observată,  $y$ , prin a măsura cât de apropiate sunt acestea. Mai mult, nu voi considera apropierea fiecărei situații, ci voi cuantifica performanța  $\hat{h}(\vec{x})$  prin a calcula media acestor greșeli. Realizez că nu am clarificat cu mult conceptul ci doar am schimbat terminologia, dar prin această modificare ne va fi mult mai ușor să decidem asupra unei metrici calculabile prin care să evaluăm performanța unei mapări. Această modificare de perspectivă, de la eroare (greșeală) la apropierea medie dintre eticheta observată și cea prezisă de  $\hat{h}(\vec{x})$  ne permite să folosim funcții care măsoară distanța dintre două puncte pentru a evalua maparea.

Cum arată o funcție de distanță? În principiu, putem avea multe alternative deoarece orice funcție care îndeplinește anumite criterii poate fi considerată o funcție ce măsoară distanța dintre două obiecte<sup>34</sup> (Deisenroth, Faisal, & Ong, 2020).

În continuare imaginați-vă că  $y$  și  $\hat{y}$  sunt defapt puncte într-un spațiu multidimensional (în exemplele de mai jos vom vedea ilustrat acest concept pentru o situație în două dimensiuni). Aș putea măsura cât de aproape este  $y_i$  de  $\hat{y}_i$  prin următoarea metrică<sup>35</sup>:

$$E_i = y_i - \hat{y}_i = -(\hat{y}_i - y_i) \text{ [E4]}$$

Folosind această definiție a distanței dintre predicția făcută de maparea căutată și valoarea observată, putem calcula cât de bună este aceasta prin a calcula distanța medie (sau eroarea medie) folosind ecuația [E5].

$$E = \frac{1}{m} \sum_1^m (y_i - \hat{y}_i) \text{ [E5]}$$

Definiția din ecuația [E5] este o măsură de distanță validă, însă nu e neapărat cea mai eficientă variantă pentru a cuantifica performanța mapării noastre deoarece suntem interesați de valoarea medie a distanței (media pentru toate observațiile). Metrica de mai sus nu este robustă în această situație deoarece pot exista cazuri în care maparea să facă greșeli la fel de mari doar ca în direcția opusă, ceea ce poate conduce la o distanță medie egală cu 0, prin urmare la o performanță foarte bună (puteți vedea un exemplu în figura 10) (de asemenea puteți vedea codul cu ajutorul căruia a fost generată această imagine în Exemplul de cod 26).

<sup>34</sup> Observați că nu am denumit observațiile puncte deoarece funcțiile de distanță nu sunt raportate doar la distanța “spațială” dintre obiecte.

<sup>35</sup> Când evaluăm performanța unei mapări dorim să obținem o distanță medie cât mai mică, prin urmare o apropiere cât mai mare dintre predicții și realitate.

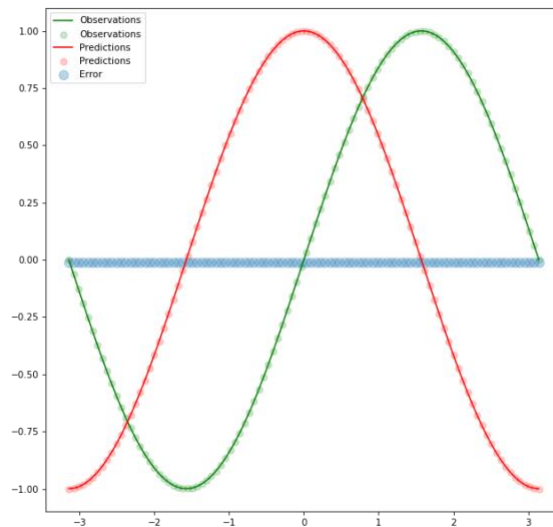


Figura 10. Exemplu unei situații în care folosind metrica din ecuația [E4], o mapare poate avea distanța medie zero chiar dacă aceasta greșește în mod frecvent să prezică valorile corecte. În figura sunt reprezentate observațiile originale (punctele și linia verzi), predicțiile făcute de o potențială mapare (punctele și linia roșii) și eroarea medie (punctele albastre).

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

x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)
y_hat = np.cos(x)

mean_distance = np.mean(y_hat - y)

_ = plt.figure(figsize=(10, 10), dpi=75)
plt.plot(x, y, c='g')
plt.scatter(x, y, c='g', s=50, alpha=0.2)
plt.plot(x, y_hat, c='r')
plt.scatter(x, y_hat, c='r', s=50, alpha=0.2)
plt.scatter(x, [mean_distance for _ in range(len(x))], s=100, alpha=0.3)
legend = plt.legend(['Observations', 'Observations',
                    'Predictions', 'Predictions', 'Error'])
plt.savefig('sample-distance-supervised-learning.png')
plt.show()
```

Exemplul de cod 26. Exemplul unde eroarea definită folosind E5 poate genera o eroare medie 0 ceea ce poate conduce la o evaluare greșită a performanței unei mapări.

Aceste situații se pot întâmpla foarte frecvent în practică. De aceea suntem nevoiți să revizuim metrica pe care o folosim pentru a evalua  $\hat{h}(\vec{x})$ . Cele mai ușoare modificări pe care le putem aduce sunt:

1. Modificăm definiția distanței dintre predicție și valoarea observată prin a considera valoarea absolută a acestei diferențe (ecuația [E6]).

$$E_i = |y_i - \hat{y}_i| \text{ [E6]}$$

2. Modificăm definiția distanței dintre predicție și valoarea observată prin a considera valoarea ridicată la pătrat a acestei diferențe, echivalentul distanței dintre două puncte în plan (ecuația [E7]).

$$E_i = (y_i - \hat{y}_i)^2 \text{ [E7]}$$

În continuare vom analiza ce efecte au fiecare dintre modificările aduse mai sus. Prima dată ne vom uita la ecuația [E6]. Luând în calcul doar valoarea absolută a erorii, performanța unei mapări poate fi dată de următoarea ecuație:

$$E = \frac{1}{m} \sum_1^m |y_i - \hat{y}_i| \text{ [E8]}$$

Similar, luând în calcul valoarea pătratului diferenței dintre predicție și realitate, vom ajunge să evaluăm performanța unei mapări folosind ecuația [E9].

$$E = \sqrt{\frac{1}{m} \sum_1^m (y_i - \hat{y}_i)^2} \text{ [E9]}$$

Cele două ecuații, ecuațiile [E8] și [E9], sunt asemănătoare. Ambele transformă diferența dintre predicție și valoarea reală într-un număr pozitiv și ambele calculează media acestor valori. În plus, ambele variază de la zero, maparea ideală, la infinit, valoarea cea mai puțin ideală și ambele penalizează diferențele mari mai mult decât diferențele mici.

Dacă sunt atât de asemănătoare, ce avantaje avem prin a folosi două metrici diferite pentru a stabili cât de eficientă este o mapare? Cheia stă în modul în care aceste metrici tratează erorile mari. Folosind metrica [E9] vom penaliza mapările care fac greșeli mari (chiar dacă doar pentru câteva observații) mult mai mult decât în cazul folosirii metricii [E8]. Cu alte cuvinte, metrica [E9] crește în funcție de variația distribuției magnitudinilor erorilor individuale, cu cât maparea face mai multe erori „grave” cu atât aceasta metrică o va penaliza (Botchkarev, 2019).

O altă diferență dintre cele două metrici este că folosind metrica [E9] vom observa că aceasta este mult mai mare decât valoarea oferită de metrica [E8] pe măsură ce numărul de observații crește (Botchkarev, 2019).



	x	y	y_hat	mae	rmse
0	-1.0	1.00	1.778407	0.778407	0.605917
1	-0.5	0.25	2.826643	2.576643	6.639091
2	0.0	0.00	-0.891271	0.891271	0.794364
3	0.5	0.25	0.300250	0.050250	0.002525
4	1.0	1.00	-0.903902	1.903902	3.624845

Figura 11. MAE si RMSE pentru o mapare aleatorie.

În continuare vom analiza un exemplu concret pentru a putea înțelege mai bine ce încearcă să spună ecuațiile [E6] – [E10]. În figura 11 avem erorile făcute de o mapare folosind ecuația [E8], respectiv ecuația [E9]. Din câte putem vedea, erorile mari sunt mult mai semnificative când folosim ecuația [E9] decât atunci când folosim ecuația [E8] (vezi rândul cu indexul 1 din figura 11). În plus, valoarea erorii totale este de 1.527 când folosim ecuația [E9] comparativ cu 1.24 când folosim ecuația [E8]. Mai mult, în figura 12 putem vedea dintr-o altă perspectivă cum ecuația [E9] penalizează mult mai mult greșelile mari, diminuând în același timp impactul erorilor foarte mici prin comparație cu ecuația [E8].

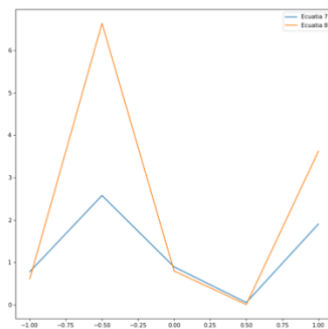


Figura 12. Reprezentare grafică a erorilor calculate folosind ecuația 7 (linia albastră) și ecuația 8 (linia portocalie). Din aceasta figură putem vedea că ecuația 8 accentuează diferențele mari dintre predicții și realitate și diminuează puțin diferențele mici dintre acestea.

Dacă doriți să investigați aceste date, puteți folosi codul din Exemplul de cod 27. Datele folosite pentru acest exemplu au fost generate fără o structură anume în minte. În acest exemplu aveți ocazia să vedeți cum putem folosi NumPy și Pandas.

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

x = np.linspace(-1, 1, 5)
y = x * x
y_hat = 4 * np.random.random_sample((len(x),)) - 1

mae = np.abs(y_hat - y)
rmse = (y_hat - y)** 2
```

```
df = pd.DataFrame(columns=["x", "y", "y_hat", "mae", "rmse"])
df["x"] = x
df["y"] = y
df["y_hat"] = y_hat
df["mae"] = mae
df["rmse"] = rmse

df.to_csv('mae-vs-rmse.csv')
```

Exemplul de cod 27. Exemplificarea diferențelor dintre MAE și RMSE.

În concluzie, prin manipularea definiției apropierii dintre valoarea prezisă de maparea noastră și valoarea observată putem să construim o metrică robustă pentru a evalua performanța acestora. Mai sus am văzut două variante prin care putem stabili cât de aproape este o mapare de realitatea observată (ecuațiile [E9] și [E9]). Aceste metrici nu sunt singurele disponibile, însă sunt unele dintre cele mai folosite pentru probleme de regresie. Există mult mai multe metrici pe care le puteți găsi în literatură sau consultând (Botchkarev, 2019).

Până acum am discutat doar despre regresie. În cele ce urmează vom vedea cum putem aplica aceleași concepte și pentru problemele de clasificare (în particular pentru sarcinile în care trebuie să alegem din două valori posibile, așa numitele clasificări binare). Similar cu situația regresiei, și într-o problemă de clasificare avem următoarea situație:

1. Avem un set de date, de exemplu,  $\{(\vec{x}_i, y_i)\}, i = \{1, 2, 3, \dots, m\}$
2. Știm, sau mai bine spus, presupunem că există o mapare dintre observații,  $\mathbf{x}$ , și etichete,  $y$ . Vom nota această mapare cu  $h(\vec{x})$ , pentru orice valoare  $\mathbf{x}$
3. Maparea ne este necunoscută, dar știm că pentru anumite valori ale lui  $\mathbf{x}_i$ , aceasta are valoarea  $y_i$  (avem această informație din setul nostru de date etichetate)

Principala diferență este că în cazul unei clasificări binare, eticheta poate avea doar două valori posibile, de exemplu, 0 și 1. În această situație, ideea de distanță sau de apropiere dintre predicția făcută de maparea aleasă și eticheta observată nu are aceeași interpretare. Dar, putem folosi aceleași ecuații, [E6] – [E9] pentru a cuantifica cât de bună este o mapare, atât timp cât ne asigurăm că înțelegem interpretarea acestora.

Studiind ecuațiile [E6] – [E9] observăm că pentru o problema de clasificare, MAE și RMSE au aceeași valoare deoarece  $y_i$  poate avea valoarea 0 sau 1

$$|y_i - \hat{y}_i| = (y_i - \hat{y}_i)^2 \text{ [E10]}$$

Prin urmare, eroarea totală este aceeași indiferent de formula pe care o folosim și reprezintă procentul de observații etichetate corect dintre toate observațiile disponibile (acest număr este

cunoscut și că acuratețea mapării). În figura 13 putem vedea un exemplu de clasificare binară în care am ilustrat conceptele de mai sus.

	x	y	y_hat	mae	rmse
0	-1.0	0	1	1	1
1	-0.5	0	1	1	1
2	0.0	1	1	0	0
3	0.5	1	1	0	0
4	1.0	1	1	0	0

Figura 13. MAE si RMSE pentru o clasificare (mapare) aleatorie.

Ca în cazul regresiei, și pentru clasificare avem multe metrici prin care să evaluăm performanța unei mapări, dar acestea nu fac parte din scopul acestei lucrări (puteți găsi mai multe metrici consultând referința (Czako, 2022)).

Cred că e momentul oportun pentru o recapitulare, să ne regroupăm puțin gândurile și să evaluăm cât de departe am ajuns.

1. Avem un set de date, de exemplu,  $\{(\vec{x}_i, y_i)\}, i = \{1, 2, 3, \dots, m\}$
2. Știm, sau mai bine spus, presupunem că există o mapare dintre observații,  $\mathbf{x}$ , și etichete,  $y$ . Vom nota aceasta mapare cu  $h(\vec{x})$ , pentru orice valoare  $\mathbf{x}$
3. Maparea ne este necunoscută, dar știm că pentru anumite valori ale lui  $\mathbf{x}_i$ , aceasta are valoarea  $y_i$  (avem aceasta informație din setul nostru de date)
4. Avem o metrică prin care putem să evaluăm cât de bună este o anumită mapare dintre observații și etichete.

Ce altceva mai trebuie să facem? Mai avem un ultim pas, și anume, să găsim o metoda prin care să descoperim aceasta mapare. Până acum am presupus că ea există și ne-am axat pe a evalua cât de bună este aceasta. Însă, în practică, nu cunoaștem maparea dintre măsurători și etichete.

În contextul învățării supravegheate, „a învăța” înseamnă să descoperim în mod automat cea mai bună mapare dintre un set de observații și etichetele acestora.

Așa cum ne-am obișnuit, o problemă bine definită, este pe jumătate rezolvată. În cele ce urmează vom proceda la fel și pentru a ajunge la o metodă prin care să aflăm cea mai bună mapare,  $\hat{h}(\vec{x})$  folosind observațiile etichete.

Primul lucru pe care trebuie să îl facem este să stabilim ce înțelegem prin cea mai bună mapare. În contextul nostru, putem presupune că cea mai bună mapare este aceea care înregistrează cea mai mică eroare,  $E$ , pentru toate observațiile disponibile. Această eroare o putem calcula folosind una dintre metricile discutate anterior (ecuațiile [E8] sau [E10] în funcție de tipul problemei pe care o rezolvăm). Prin urmare, cea mai bună  $\hat{h}(\vec{x})$  este cea mapare care are o eroare 0. În realitate nu vom ajunge niciodată să avem o eroare zero din cauza mai multor factori precum erorile de măsurare. Așadar, sarcina noastră este să găsim o mapare,  $\hat{h}(\vec{x})$ , pentru care eroarea,  $E$ , este minimă.

O prima abordare a acestei probleme este să folosim teoria optimizării. Pentru aceasta avem nevoie să rescriem problema sub forma unei probleme de optimizare. Cea mai simplă variantă este să plecăm de la o presupunere despre forma mapării. În cazul unei regresii am putea presupune că maparea căutată are forma [E11]. De precizat că aceasta nu este singura formă posibilă a unei mapări. În practică alegem o formă a mapării în funcție de problema pe care o rezolvăm și de datele pe care le avem la dispoziție. Un studiu amănunțit al datelor sau o cunoaștere a sistemului ne poate oferi indicii asupra formei mapării căutate. La finalul capitolului vom vedea câteva tipuri de mapări cu care putem rezolva diferite probleme de regresie și clasificare.

$$\hat{h}(\vec{x}) = w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n \text{ [E11]}$$

Înarmați cu această presupunere, găsirea celei mai bune mapări devine o căutare de parametri (valorile  $w_1, w_2, \dots, w_n$  din ecuația [E11]) pentru care predicțiile făcute de  $\hat{h}(\vec{x})$  au cea mai mică eroare posibilă pentru toate observațiile disponibile.

Formal, aceasta problema se poate formula astfel:

Găsiți maparea  $\hat{h}(\vec{x})$ , unde  $\hat{h}(\vec{x}) = w_1 * x_1 + w_2 * x_2 + \dots + w_n * x_n$ , pentru care  $E = \sqrt{\frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2}$  măsurată pe o mulțime de date etichetate,  $\{(\vec{x}_i, y_i)\}, i = \{1, 2, 3, \dots, m\}$ , are cea mai mică valoare posibilă.

Întrebarea care se ivește acum este cum putem afla acei parametri într-un mod automat. O varianta naiva ar fi să încercăm mai multe valori la întâmplare până când epuizăm toate valorile posibile pentru  $w_1, w_2, \dots, w_n$ . După cum vă dați seama, această abordare nu este foarte eficientă deoarece parametrii din ecuația [E11] au valori continue ceea ce înseamnă că ar trebui să încercăm o infinitate de valori pentru a fi siguri că găsim perechea pentru care  $E$  este minimă. Din fericire, avem la dispoziție două abordări mai eficiente. Prima abordare este bazată pe gradienti iar cea de a doua nu se bazează pe gradienti. Foarte intuitiv, știu! Voi clarifica aceste aspecte imediat.

Ne vom concentra atenția asupra optimizării bazate pe gradienti. Optimizarea fără gradienti depășește scopul acestei lucrări, dar dacă vreți să aflați mai multe detalii puteți să consultați referința (Jang, Sun, & Mizutani, 1997)

Înainte de toate este nevoie să precizăm că pentru scopurile noastre, maparea căutată este o funcție. În matematică, o funcție este o relație între un set de intrări și un set de posibile ieșiri cu proprietatea că fiecare intrare este legată de exact o ieșire. Cu alte cuvinte, o funcție este o regulă care atribuie o ieșire unică fiecărei intrări.

O mapare, cunoscută și sub numele de funcție sau transformare, este un concept matematic care descrie cum să preiau elemente dintr-un set, numit domeniu, și să le atribuim elementelor dintr-un alt set, numit interval. Mapările pot fi reprezentate ca perechi ordonate, scrise ca  $(x, y)$ , unde  $x$  este un element din domeniu și  $y$  este elementul corespunzător din interval.

Revenind, optimizarea bazată pe gradienti este una dintre cele mai folosite metode pentru a afla parametrii pentru care o funcție atinge valoarea minimă sau maximă. Această metodă pleacă de la o observație că putem găsi punctul de minim local al unei funcții diferențiabile „pășind” în mod repetat în direcția opusă a gradientului calculat la un anumit punct. Acest lucru este posibil deoarece această direcție, opusă gradientului, este direcția cea mai abruptă (direcția de-a lungul căreia are loc cea mai rapidă schimbare a valorii funcției). Figura 14 prezintă un exemplu de funcție diferențiabilă și gradientul acesteia. După cate putem observa, direcția opusă gradientului corespunde direcției în care funcția manifestă cele mai rapide schimbări.

Derivata unei funcții reprezintă rata cu care această funcție variază într-un anumit punct. De exemplu, pentru o funcție de o variabilă, definită pe întreaga axă a numerelor reale, putem calcula derivata acesteia în punctul  $a$  aflând valoarea limitei de mai jos

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h} \quad [E12]$$

Această limită ne oferă informații despre cât de mult se modifică funcția  $f$  în vecinătatea punctului  $a$ . De asemenea, aceeași limită poate fi interpretată și ca panta liniei tangente la graficul funcției  $f$  în punctul  $a$ .

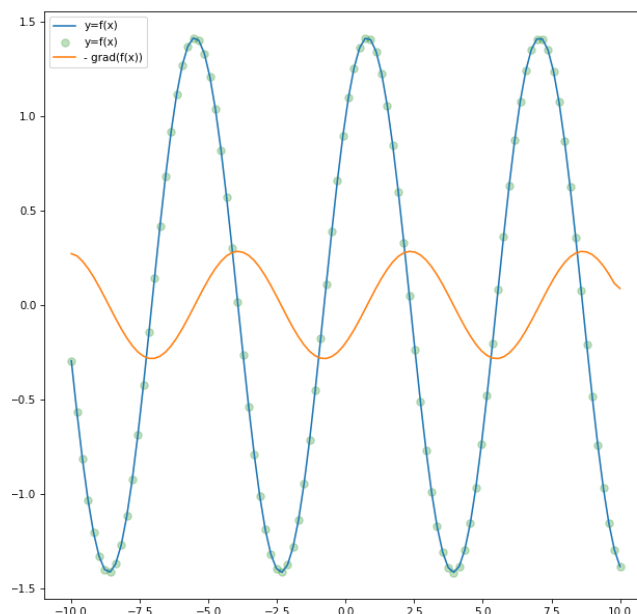


Figura 14. Graficul reprezintă funcția  $y = \sin(x) + \cos(x)$  (linia albastră și punctele verzi) și opusul gradientului acesteia (linia portocalie) calculate pentru valori ale lui  $x$  între -10 și 10. Așa cum putem observa, dacă izolăm un punct, de exemplu  $x = 2.5$ , funcția se schimbă cel mai repede de-a lungul direcției opuse gradientului în acel punct.

Același principiu îl putem aplica și pentru o funcție de mai multe variabile. Ecuația [E12] poate fi aplicată pentru fiecare variabilă, ținând restul variabilelor „constante”. Cu alte cuvinte, putem aplica ecuația [E12] să aflăm derivata acesteia de-a lungul fiecărei axe. În cazul funcțiilor de mai multe variabile, derivata în funcție de o variabilă devine o derivată parțială a funcției.

O funcție este diferențiabilă dacă putem calcula limita din ecuația [E12] pentru orice punct  $x$  din domeniul funcției (mulțimea de valori pe care este definită funcția). În cazul funcțiilor de mai multe variabile,  $f: R^n \rightarrow R$ , definim gradientul funcției ca vectorul format din derivatele parțiale ale funcției  $f$  astfel:

$$\text{grad}(f) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad [\text{E13}]$$

unde  $\frac{\partial f}{\partial x_i}$  reprezintă derivata parțială a lui  $f$  în funcție de  $x_i$ .

Este important să observăm că pentru o funcție  $f(x)$  pot exista mai multe valori ale lui  $x$  pentru care funcția are o valoare minimă sau maximă. Aceste puncte poartă denumirea colectivă de puncte de extrem. În funcție de valoarea lui  $x$  și valorile funcției din vecinătatea acestuia, ne putem regăsi în puncte de minim sau de maxim local. Cu alte cuvinte, o valoare extremă nu este neapărat extremul absolut al funcției. Este posibil ca aceasta să fie doar o valoare de extrem local (doar în vecinătatea lui  $x$ ) (vezi figura 14). Pentru a determina punctele de extrem este

suficient să studiem ce se întâmplă cu derivata funcției. Când derivata funcției este mai mare decât zero, valorile funcției cresc. Viceversa, când derivata funcției este mai mică decât zero, valorile funcției scad (vezi definiția din [E12]).

Dacă atingem un punct de maxim, înseamnă că am atins un punct,  $a$ , în care valoarea funcției nu mai poate crește. Aceasta înseamnă că pentru orice punct  $x$  la dreapta lui  $a$ , valoarea funcției scade ceea ce înseamnă că derivata funcției este negativă. Deoarece trecem de la o creștere la o descreștere a valorilor funcției sau de la o derivată pozitivă la una negativă și funcția fiind continuă, înseamnă că în momentul de maxim derivata funcției trebuie să aibă valoarea zero. Același lucru se întâmplă și în cazul unui punct de minim local, cu precizarea că tranziția derivatei se face de la o valoare negativă la una pozitivă. În concluzie, punctele de extrem sunt punctele pentru care valoarea derivatei este 0.

Pentru a determina dacă aceste puncte de extrem reprezintă o valoare minimă sau maximă este suficient să aplicăm același argument pentru valoarea celei de-a doua derivate a funcției. Studiind comportamentul acestora descoperim că pentru o valoare de minim local, derivata de ordin doi este negativă iar pentru o valoare de maxim local derivata de ordin doi este pozitivă. În cazul în care derivata de ordin doi are valoarea zero, punctul de extrem este un punct de inflexiune.

Această paranteză lungă în lumea analizei matematice ne oferă uneltele necesare pentru a determina cea mai bună funcție care transformă observațiile noastre într-o etichetă. În cele ce urmează vom construi un prim algoritm de învățare bazat pe gradienti prin care să minimizăm valoarea erorii,  $E$ , pentru o anumită formă a mapării<sup>36</sup> noastre.

1. Alegem la întâmplare o serie de valori pentru parametrii  $w_1, w_2, \dots, w_n$
2. Calculăm eroarea  $E$
3. Calculăm gradientul erorii  $E$  în funcție de parametrii  $w_1, w_2, \dots, w_n$
4. Modificăm valoarea parametrilor în baza regulii de actualizare de mai jos:

$$w_{i,k+1} = w_{i,k} - \gamma * grad(E) \text{ [E14].}$$

Gradientul lui  $E$  este calculat în funcție de parametrii  $w_1, w_2, \dots, w_n$ , deoarece valorile lui  $x$  sunt considerate cunoscute și constante. Ne interesează să aflăm valorile parametrilor, valorile lui  $w$ , pentru care funcția noastră de eroare este minimă.

5. Repetăm pașii 2 – 4 până când eroarea  $E$  nu se mai modifică de la o iterație la alta sau până când  $|E_{k+1} - E_k| < e$ , unde  $e$  este o valoare aleasă la începutul algoritmului.

---

<sup>36</sup> Voi folosi mapare și funcție cu același înțeles deși tehnic vorbind există diferențe între aceste două concepte, așa cum am văzut în paragrafele anterioare.

La finalul acestei proceduri vom avea garantat una dintre cele mai bune funcții pentru a transforma observațiile în etichetele corespunzătoare pentru eșantionul nostru de date.

Observați că această metoda, deși ne garantează că va converge (găsește o funcție pentru care eroare este minimă), nu ne garantează că valoarea către care converge este și valoarea minimă globală pentru orice funcție de eroare. Este foarte probabil ca algoritmul nostru de optimizare să rămână blocat într-o regiune de minim local al funcției de eroare.

Putem aplica această procedură indiferent de problema pe care o rezolvăm (clasificare sau regresie) atât timp cât funcția de eroare este diferențiabilă pentru orice valoare a parametrilor  $w_1, w_2, \dots, w_n$ .

Am ajuns aproape de finalul metodei de învățare supravegheate, dar mai avem un singur lucru de stabilit. În introducerea acestei secțiuni am spus că vrem să găsim o mapare care să descrie regulile de funcționare ale unui sistem. Eram interesați să aflăm cum funcționează un sistem plecând de la niște observații despre acesta. Până acum nu am făcut decât să găsim o mapare care să aibă o eroare minimă pe setul de date observate. Este oare garantat că aceasta va performa la fel pe un set de date pe care nu le-a întâlnit? Din păcate nu avem această garanție.

Prin urmare, înainte de a ne declara învingători, suntem nevoiți să retestăm performanța mapării pe un set de date noi, ideal unul care conține observații pe care nu le-am folosit în determinarea mapării. Această verificare finală ne oferă o indicație asupra abilității de generalizare a mapării noastre. Dacă funcția descoperită este optimă ar trebui să avem o eroare aproximativ la fel de mică pe un set de date noi precum cea de pe setul de antrenare. Similar, în cazul clasificării, ar trebui să avem o acuratețe similară pe ambele seturi de date. În caz contrar, avem o problemă deoarece maparea noastră nu este suficient de generală ceea ce înseamnă că a învățat niște caracteristici ale datelor specifice observațiilor inițiale (acestea neregăsindu-se în setul nou de date). Acest fenomen poartă numele de supra adaptare.

În același timp este posibil ca maparea pe care am descoperit-o să facă în mod repetat greșeli deoarece forma aleasă pentru această mapare este prea simplă pentru tiparele pe care dorim să le descoperim. În acest caz spunem că avem o mapare sub adaptată pentru situația dată.

Aceste două concepte, sub și supra adaptarea, sunt strâns legate de două idei foarte importante în statistică, și anume, biasul și varianța. În cele ce urmează vom introduce cele două concepte.

## 4.2. Bias și varianță

Demaram această discuție presupunând ca avem un model,  $M$ , prin care explicăm comportamentul unui sistem. Exista oare o limitare a acestui model în privința relațiilor dintre observații și etichete pe care acesta le poate reprezenta? Și dacă exista, care este aceea? Mai



mult, ce se întâmplă dacă variem puțin valorile fiecărei observații? Vor varia și predicțiile cu același ordin de mărime sau se vor suferi modificări drastice?

Pentru a putea răspunde mai ușor la aceste întrebări, vom începe prin a distinge între două situații. În prima situație dorim să descoperim dacă modelul nostru este limitat din punct de vedere al relațiilor pe care le poate reprezenta. Cu alte cuvinte dorim să aflăm dacă modelul nostru este în vreun fel părtinitor. Ideal ar fi să putem să și cuantificăm cât de părtinitor este acesta. Formal, acest concept poartă denumirea de bias. Informal, acest bias reprezintă o exagerare incorectă în favoarea unei situații. Din punct de vedere statistic, biasul reprezintă tendința unui model sau a unui estimator de a favoriza anumite date atunci când ne adunăm observațiile ceea ce inevitabil conduce la rezultate incorecte sau chiar false. Tot din punct de vedere statistic, avem o a treia interpretare a acestui concept (și cea care ne interesează în acest capitol) ca și eroarea sistematică pe care modelul nostru o face în timpul predicțiilor. În acest context, un model cu un bias mare are tendința să facă predicții care sunt prea mari sau prea mici comparativ cu valoarea reală (James, Witten, Hastie & Tibshirani, 2017) (Bishop, 2006) (Murphy, 2012).

De asemenea, un model care manifestă un bias mare va conduce de cele mai multe ori la sub adaptare (underfit) deoarece acesta nu va fi capabil să găsească tiparele potrivite pentru a conecta observațiile de etichete. În figura 15 regăsim două situații. Prima (imaginea de sus) reprezintă un model care are un bias mare iar cea de-a doua reprezintă un model care are un bias mic. După cum putem observa, în prima situație, predicția făcută de acest model este complet greșită, acesta prezentând o eroare foarte mare. În schimb, modelul cu bias mic (imaginea de jos), reușește să prezică valoarea reală cu o eroare foarte mică.

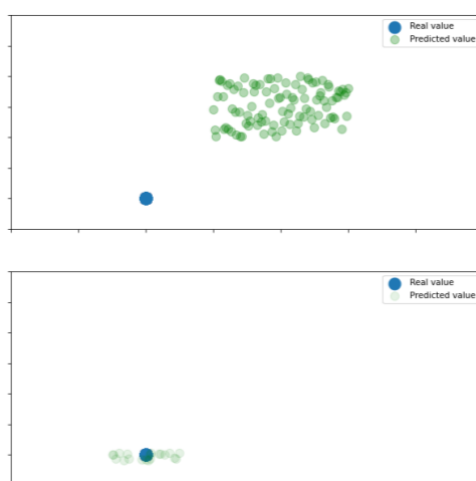


Figura 15. Exemplul unei estimări cu bias mare (graficul de sus) și al unei estimări cu bias mic (graficul de jos)

Dacă în loc de o singură valoare am vrea acum să prezicem etichetele pentru întreg setul de date (datele folosite pentru antrenare cât și cele folosite pentru testare) observăm că modelul

nu reușește nici pe departe să reprezinte structura datelor. Mai mult, acesta găsește o relație incorectă între observații și etichete (vezi figura 16).

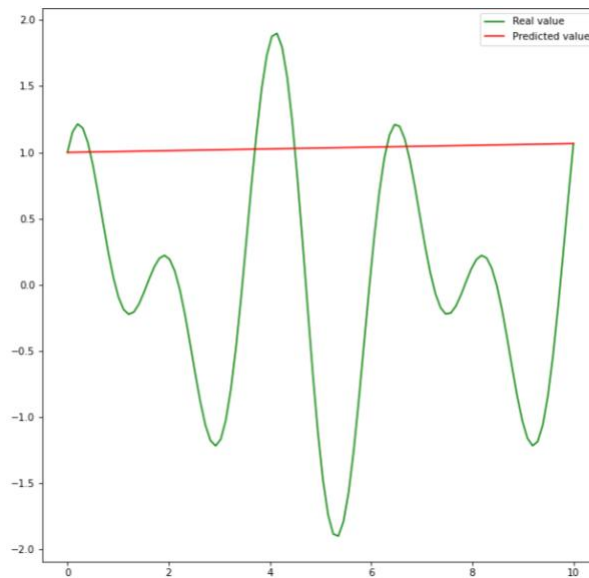


Figura 16. Exemplul unui model care are o eroare sistematică mare. Observați cum modelul nu reușește să se apropie aproape deloc de valorile observate. Mai mult, modelul nu reușește să descopere decât o relație liniară,  $y = ax + b$  între observații și etichete. Această relație este departe de relația reală,  $y = \sin 2x + \cos 3x$

În a doua situație, ne interesează să aflăm cât de susceptibil este modelul nostru atunci când valorile de intrare sau observațiile variază puțin. Cu alte cuvinte, ne interesează să aflăm dacă mici modificări ale valorilor observațiilor conduc la modificări majore ale valorilor predicțiilor. Formal, putem defini acest concept drept varianța modelului sau a estimatorului. Varianța este o cantitate prin care putem determina cât de „împrăștiate” sunt valorile dintr-un set de date și reprezintă media deviațiilor pătratice a valorilor unei componente a unei observații de la media acesteia (Murphy, 2012) (Bishop, 2006) (James, Witten, Hastie, & Tibshirani, 2017). Putem calcula varianța folosind una din ecuațiile [E15] sau [E17].

$$Var(X) = E[X - \mu] \text{ [E15]}$$

unde  $\mu$  reprezintă media (calculată cu formula din ecuația [E16]) iar  $E$  reprezintă valoarea așteptată a acestei diferențe.

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i \text{ [E16]}$$

În cazul în care fiecare dintre valorile componente  $x$  au aceeași șansă de a apărea în observațiile noastre, putem înlocui ecuația [E15] cu ecuația [E17]

$$Var(X) = \frac{1}{n} \sum_{i=1}^m (x_i - \mu)^2 \text{ [E17]}$$

unde  $\mu$  este media valorilor componente la fel ca în cazul ecuației [E16].

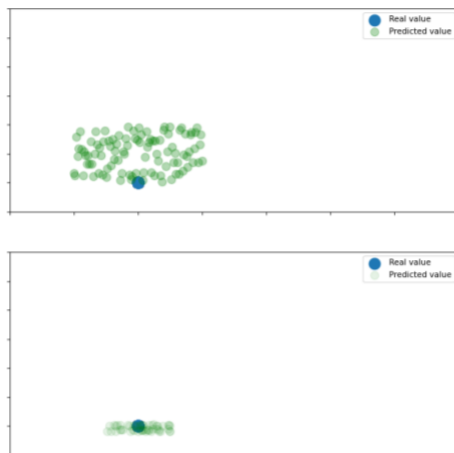


Figura 17. Exemplul unei estimări cu varianta mare (graficul de sus) și al unei estimări cu varianta mic (graficul de jos)

În cazul modelelor sau estimatorilor cu varianta mare, vom observa că predicțiile pe care acestea le fac variază foarte mult pentru modificări foarte mici ale datelor de intrare (vezi figura 17). Mai mult, când modelul are o varianta mare, este foarte ușor pentru acesta să „memoreze” observațiile folosite în timpul antrenării acestuia. Modelul astfel rezultat având o performanță foarte slabă atunci când este folosit cu date noi, pe care nu le-a mai văzut în timpul antrenării deoarece el este supra adaptat (overfit) la tiparele întâlnite în setul de antrenament (vezi figura 18).

Mai avem un ultim caz de acoperit înainte de a încheia discuția noastră despre bias și varianță. Poate un model să aibă o eroare sistematică mare și o varianță mare simultan? În realitate, da. Cele două situații prezentate mai sus nu se exclud. Există modele care pot avea:

1. Bias mare și varianță mică
2. Bias mic și varianță mare
3. Bias mare și varianță mare
4. Bias mic și varianță mică

De aceea, în practică, trebuie să fim atenți în alegerea modelului. Ideal ar fi să avem doar modele care au bias mic și varianță mică, dar acest lucru nu este tot timpul posibil. Prin urmare, când alegem un model trebuie să balansăm biasul și varianța pe care acesta le are. Această balanșare poartă denumirea de „compromisul dintre bias și varianță”. Mai mult, când alegem o mapare care are un bias mare sau o varianță mare, nu înseamnă că trebuie să abandonăm

complet problema. Există o suită de metode pe care le putem folosi pentru a diminua efectele biasului și ale varianței.

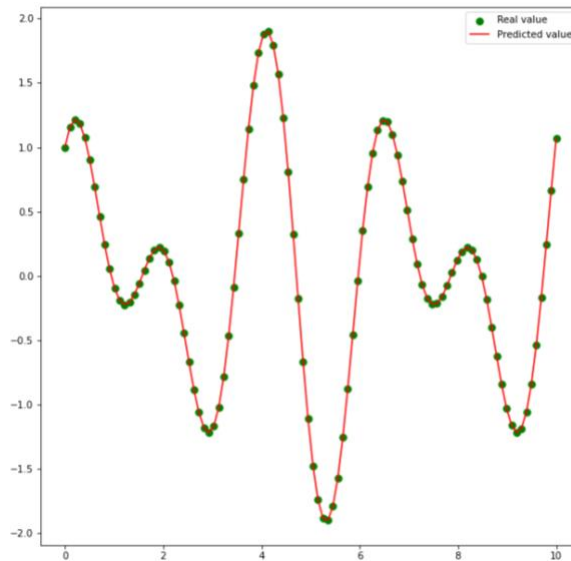


Figura 18. Exemplul unui model supra adaptat (linia roșie). Observați cum modelul reușește memoreze toate valorile observate (punctele verzi).

După această introducere informală a biasului și a varianței, vom alocă restul acestei secțiuni pentru a trata matematic cele două concepte. Presupunem că dorim să învățăm o funcție,  $h(x)$  folosind un set de observații  $y = h(x) + \varepsilon$ , unde  $\varepsilon$  reprezintă eroarea totală de măsurare; aceasta are media 0 și deviația standard diferită de zero. Mai mult, pentru a învața (aproxima) funcția  $h(x)$ , vom folosi o funcție de regresie,  $\hat{h}(x)$ .

Presupunem că am descoperit deja parametrii funcției  $\hat{h}(x)$  și o folosim pentru a face o predicție. Dată fiind o valoare nouă,  $x_k$ , eroarea modelului nostru va fi:

$$E \left[ \left( y - \hat{h}(x) \right)^2 \middle| x = x_0 \right] \quad [\text{E18}]$$

Prin mai multe manipulări algebrice, putem reduce expresia [E18] la următoarea formă:

$$E \left[ \left( y - \hat{h}(x) \right)^2 \middle| x = x_0 \right] = \sigma_\varepsilon^2 + \left( E[\hat{h}(x_0)] - f(x_0) \right)^2 + E \left[ \hat{h}(x_0) - E[\hat{h}(x_0)] \right]^2 \quad [\text{E19}]$$

Termenii din ecuația [E16] ne oferă următoarele informații:

- $\sigma_\varepsilon^2$  este deviația standard a erorii de măsurare.

- $\left(E[\hat{h}(x_0)] - f(x_0)\right)^2$  reprezintă eroarea dintre valoarea așteptată a predicției și valoarea reală a funcției pentru  $x_0$ . Această cantitate reprezintă biasul modelului nostru.
- $E\left[\hat{h}(x_0) - E[\hat{h}(x_0)]\right]^2$  reprezintă varianța modelului.

Gândiți-vă acum că avem un model foarte complex (de exemplu, un model cu foarte mulți parametri, de forma unui polinom) pe care îl „potrivim” pe setul de observații. Deoarece acest model are foarte mulți parametri, este foarte ușor pentru el să se adapteze la datele de antrenare astfel încât să uniformizeze media. În urma acestei potriviri modelul vă ajunge să aibă un bias foarte redus (el reușește să prezică fiecare observație). În același timp, deoarece avem un număr foarte mare de parametri, este foarte probabil că diferențele pătratice să fie foarte variate. Având o varianță mare, acest model vă avea o performanță proastă când întâlnește date noi (spunem că el nu generalizează). Acest lucru se întâmplă deoarece erorile mici sunt cumulate într-o eroare finală foarte mare (având foarte mulți parametri, vom însuma multe variații mici).

Avem și situația contrară unde modelul ales are foarte puțini parametri. În acest caz erorile generate nu vor fi foarte mari (deoarece avem de-a face cu puțini parametri). În schimb, estimarea pe care modelul o face nu vă corespunde îndeaproape cu valoarea reală a etichetei. Prin urmare, acest model vă avea un bias foarte ridicat.

În practică, mapările care au un bias ridicat vor genera o eroare mare pe setul de antrenare și pe setul de test. Contrar, valorile care au o varianță foarte mare pot genera o eroare foarte mică, uneori chiar zero, pe setul de antrenare și o eroare mare pe setul de test. Prin urmare, când dezvoltăm soluții de învățare trebuie să fim atenți în permanență la „schimbul” dintre bias și varianță deoarece nu vom reuși niciodată să avem o mapare ideală.

În concluzie, învățarea supravegheată ne oferă o serie de metode prin intermediul cărora putem construi o mașină care să învețe cum să rezolve o anumită problemă sau să ia o anumită decizie fără programăm în mod specific ce reguli trebuie să urmeze. Mai mult, folosindu-ne de învățarea supravegheată putem să descoperim regulile după care se desfășoară un fenomen sau un proces plecând doar de la observații etichetate ale acestora. În finalul acestei capitole vom vedea algoritmi din învățarea supravegheată cu ajutorul cărora putem rezolva probleme de regresie sau de clasificare. Ne vom opri asupra K-Nearest Neighbors (kNN), regresiei liniare, regresiei logistice și a arborilor decizionali de regresie și de clasificare.

### 4.3. K-Nearest Neighbors (kNN)

K-Nearest Neighbors este unul dintre primii algoritmi din învățarea supravegheată. Cu ajutorul acestui algoritm putem rezolva atât probleme de regresie cât și probleme de clasificare<sup>37</sup>. Ideea

<sup>37</sup> Puteti gasi mai multe detalii despre kNN aici: <https://towardsdatascience.com/the-basics-knn-for-classification-and-regression-c1e8a6c955>

din spatele algoritmului este foarte intuitivă. În cele ce urmează, vom analiza algoritmul kNN folosind o problemă de clasificare. În figura 19 vedem un eșantion cu observații pentru trei tipuri de vinuri<sup>38</sup>. Eșantionul conține mai multe caracteristici ale vinurilor, dar pentru simplitate am ales să folosim doar două, conținutul de alcool și de acid malic.

După câte putem observa, cele trei tipuri de vinuri sunt preponderent distribuite în anumite regiuni ale planului 2D format din valorile celor două componente ale unei măsurători. Plecând de la aceasta observație, am putea să ne gândim că atunci când avem o nouă observație pentru care măsurăm conținutul de acid malic și de alcool, aceasta se va regăsi undeva în acest plan. Dacă observația cea nouă este în aria punctelor galbene (partea de sus a figurii) am putea spune cu destulă certitudine că acel vin face parte din clasa vinurilor „galbene”. Similar și pentru celelalte zone. Dar ce se întâmplă atunci când noua măsurătoare ne oferă un punct la intersecția dintre cele trei zone? În această situație am putea să comparăm noua observație cu observațiile cunoscute și să luăm o decizie în baza numărului de puncte comune cu fiecare clasă. De exemplu, dacă avem un vin pentru care avem doi vecini galbeni, trei vecini verzi și 5 vecini mov, atunci concluzionăm că vinul cel nou face parte din categoria „mov”.

Dar cum determinăm care sunt vecinii unui punct? Cea mai ușoară metodă este să verificăm cât de aproape este un punct nou față de toate celelalte puncte despre care avem informații. După ce determinăm distanța dintre perechi, le ordonăm crescător, de la cele mai apropiate la cele mai îndepărtate. Ulterior extragem un număr de  $k$  puncte de la începutul acestei liste și numărăm câte exemple din fiecare clasă avem între aceste  $k$  puncte. În final, luăm o decizie în baza clasei majoritare dintre cele  $k$  puncte (clasa cu cele mai multe exemple între cele  $k$  puncte).

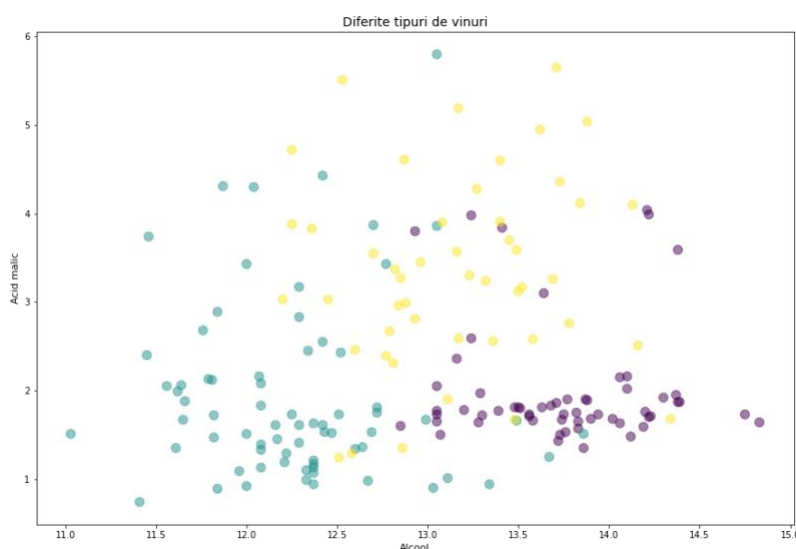


Figura 19. Trei tipuri de vinuri distribuite în funcție de conținutul de acid malic și de alcool.

```
from sklearn.datasets import load_wine
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

<sup>38</sup> Gasiti detaliile despre esantion aici: <https://archive.ics.uci.edu/ml/datasets/wine>

```

import pandas as pd

X, y = load_wine(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

results = []
K = [1, 3, 5, 8, 20]
for k in K:
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train, y_train)

    y_train_predicted = clf.predict(X_train)
    y_test_predicted = clf.predict(X_test)

    train_accuracy = accuracy_score(y_true=y_train, y_pred=y_train_predicted)
    test_accuracy = accuracy_score(y_true=y_test, y_pred=y_test_predicted)

    results.append({"k": k, "Train accuracy": train_accuracy * 100, "Test accuracy": test_accuracy * 100})

results = pd.DataFrame(results, columns=results[0].keys())
print(results)

>> k Train accuracy Test accuracy
>> 0 1 100.000000 77.777778
>> 1 3 85.915493 80.555556
>> 2 5 75.352113 72.222222
>> 3 8 79.577465 72.222222
>> 4 20 73.239437 77.777778

```

Exemplul de cod 28. Implementarea unui clasificator kNN pentru mai multe valori ale lui  $k$ .

După cum putea vedea în exemplul de cod 28, numărul de vecini influențează foarte mult acuratețea obținută pentru setul de antrenare respectiv de test.. Atunci când  $k = 1$  modelul nostru reușește să învețe întreg setul de antrenare (intrând într-un regim de supra antrenare). Pe măsură ce creștem numărul de vecini, observăm că algoritmul este din ce în ce mai confuz.

Eficiența algoritmului kNN scade pe măsură ce numărul de observații folosite pentru antrenare crește. Acest lucru se datorează numărului mare de computații pe care trebuie să îl facă. Reamintiți-vă că acest algoritm calculează distanța dintre un punct nou și toate celelalte puncte cunoscute, ordonează perechile în funcție de această distanță și la final ia o decizie. În schimb, pentru seturi de date mici, kNN poate fi un punct de plecare foarte bun pentru soluțiile noastre.

Ideea rămâne aceeași și pentru o problemă de regresie. Singura modificare pe care o facem este metoda prin care facem o predicție. De exemplu, prezicem o valoare egală cu media valorilor tuturor celor  $k$  vecini. În rest, totul rămâne neschimbat.

Înainte de a vedea și alți algoritmi, trebuie să mai zăbovim puțin asupra unei idei pe care am tratat-o cu superficialitate. Am văzut cum kNN determină cei  $k$  vecini în funcție de distanța dintre puncte. Însă, ce este aceasta distanță?

Distanța dintre două puncte poate avea mai multe definiții, în funcție de situația în care ne aflăm. Cea mai des întâlnită definiție este distanța euclidiană, diagonală hipercubului format de coordonatele  $(x_{i1}, x_{i2}, \dots, x_{in})$  ale punctului nou și coordonatele  $(x_{j1}, x_{j2}, \dots, x_{jn})$  ale unui punct existent. Aceasta distanță poate fi calculată folosind formula din ecuația [E20]

$$d_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{in} - x_{jn})^2} \text{ [E20]}$$

O altă variantă este să considerăm distanța dintre puncte ca fiind suma lungimilor laturilor parcurse să ajunge de la un punct la celălalt. Aceasta poate fi reprezentată folosind formula [E21].

$$d_{ij} = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{in} - x_{jn}| \text{ [E21]}$$

Cele doua formulări ale distanței dintre puncte nu sunt exhaustive. Dacă doriți să aflați mai multe despre alte modalități prin care putem să calculăm distanța dintre două puncte, puteți să vizitați: <https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa>.

În concluzie, kNN este un algoritm destul de eficient atunci când avem o mulțime cu un număr relativ mic de observații. Acest algoritm putând fi folosit ca o primă aproximare a soluției, fiind intuitiv și ușor de implementat. Din păcate, performanța algoritmului scade atunci când avem un eșantion mare sau când observațiile nu sunt foarte clar delimitate în spațiu.

## 4.4. Regresia liniara

Regresia liniară este posibil cel mai întâlnit (și abuzat) algoritm în problemele de regresie. După cum îi spune și numele, acest algoritm încearcă să găsească o mapare (funcție) liniară capabilă să prezică etichetele cu o eroare pătratică medie cât mai mică posibil. Intuitiv, pentru situația în care avem observații cu o singură componentă,  $x$ , vă puteți gândi la regresia liniară ca la un algoritm cu ajutorul căruia găsim dreapta care minimizează cel mai mult eroarea de predicție (vezi figura 20).

Pentru a găsi aceasta funcție liniară, folosim metoda de optimizare bazată pe gradienti prezentată în secțiunea anterioară.

1. Vrem să găsim cea mai bună funcție,  $f(\vec{x}) = w_0x_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n$ , unde  $x_0 = 1$  și  $x_1, x_2, \dots, x_n$  sunt cele  $n$  componente ale unei singure observații.
2. Alegem la întâmplare o serie de valori pentru parametrii  $w_1, w_2, \dots, w_n$
3. Calculăm eroarea  $E$



4. Calculăm gradientul erorii  $E$  în funcție de parametrii  $w_1, w_2, \dots, w_n$
5. Modificăm valoarea parametrilor folosind regula de mai jos:  

$$w_{i,k+1} = w_{i,k} - \gamma * grad(E) \text{ [E19]}$$
6. Repetăm pașii 3 – 5 până când eroarea  $E$  nu se mai modifică.

În practică, regresia liniară poate avea o varianță foarte mare dacă avem foarte mulți parametri. În același timp, dacă avem prea puțini parametri sau dacă relația dintre observații și etichete nu este liniară, regresia liniară are un bias ridicat (vezi secțiunea anterioară despre bias și varianță pentru mai multe detalii).

Pentru a remedia aceste efecte negative, putem modifica eroare astfel încât să penalizăm pentru valori extreme ale parametrilor. Găsiți mai multe detalii despre cum realizăm acest lucru în (Bishop, 2006) (Murphy, 2012) (Shalev-Shwartz & Ben-David, 2014).

În finalul acestei secțiuni regăsim două exemple de regresie liniară<sup>39</sup> implementate în Python folosind biblioteca *scikit-learn*. În aceste exemple încercăm să învățăm o mapare liniară pentru un set sintetic de date generat cu funcția *make\_regression()* oferită de biblioteca *scikit-learn*. Exemplul de cod 29 reprezintă o implementare pentru o situație 1D, iar exemplul de cod 30 reprezintă o implementare pentru o situație 2D. Observați că în ambele cazuri, regresia liniară încearcă să găsească linia sau planul care „trece” prin valorile observate cu o eroare cât mai mică. Acest cod se poate extinde și pentru situațiile în care observațiile au mai multe componente. Din păcate, în aceste situații este destul de dificil să vizualizăm rezultatele sub formă grafică.

```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

# initializeaza un set de date 1D pentru regresie
X, y = make_regression(100, 1)

# imparte datele in doua seturi (80/20) pentru antrenare si testare
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# initializeaza o regresie liniara
model = LinearRegression()

# antreneaza o regresie liniara folosind datele de test
model.fit(X_train, y_train)

# prezice valorile observatiilor pentru setul de antrenare si testare
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

---

<sup>39</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LinearRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)

```

# calculeaza MAE (eroarea medie absoluta) si MSE (eroarea medie patratica)
mae_train = mean_absolute_error(y_true=y_train, y_pred=y_train_pred)
mae_test = mean_absolute_error(y_true=y_train, y_pred=y_train_pred)

mse_train = mean_squared_error(y_true=y_test, y_pred=y_test_pred)
mse_test = mean_squared_error(y_true=y_test, y_pred=y_test_pred)

print(f"Eroare medie absoluta pentru datele de antranare = {mae_train}")
>> Eroare medie absoluta pentru datele de antranare = 3.2288061113661114e-14

print(f"Eroare medie absoluta pentru datele de test = {mae_test}")
>> Eroare medie absoluta pentru datele de test = 3.2288061113661114e-14

print(f"Eroare medie patratica pentru datele de antranare = {mse_train}")
>> Eroare medie patratica pentru datele de antranare = 1.0883815301721147e-27

print(f"Eroare medie patratica pentru datele de test = {mse_test}")
>> Eroare medie patratica pentru datele de test = 1.0883815301721147e-27

# vizualizeaza rezultate
fig, axs = plt.subplots(2, figsize=(10, 10), dpi=75)
axs[0].scatter(X_train, y_train, c='g', s=100)
axs[0].plot(X_train, y_train_pred, c='r')
axs[0].legend(['[Train] Real value', '[Train] Predicted value'])
axs[0].set_xlabel("x")
axs[0].set_ylabel("y")

axs[1].scatter(X_test, y_test, c='g', s=100)
axs[1].plot(X_test, y_test_pred, c='r')
axs[1].legend(['[Test] Real value', '[Test] Predicted value'])
axs[1].set_xlabel("x")
axs[1].set_ylabel("y")

plt.show()

```

Exemplul de cod 29. Implementarea regresiei liniare in Python folosind scikit-learn și matplotlib.

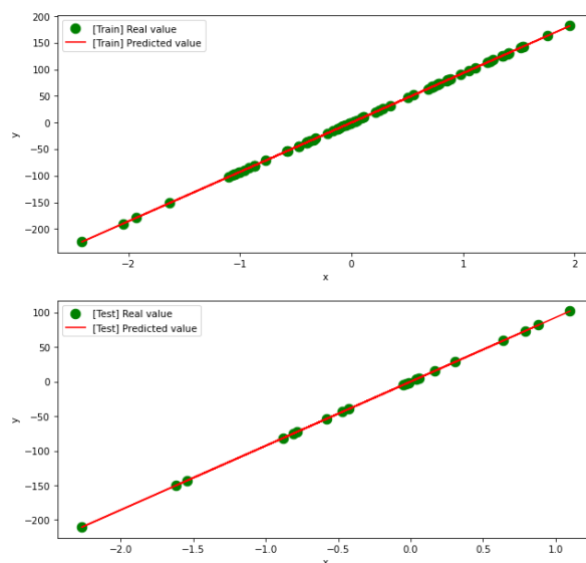


Figura 20. Rezultatele unei regresii liniare. Graficul de sus contine predictiile facute cu acest model pentru datele de antrenament. Graficul de jos contine predictiile facute de acest model pentru datele de test.

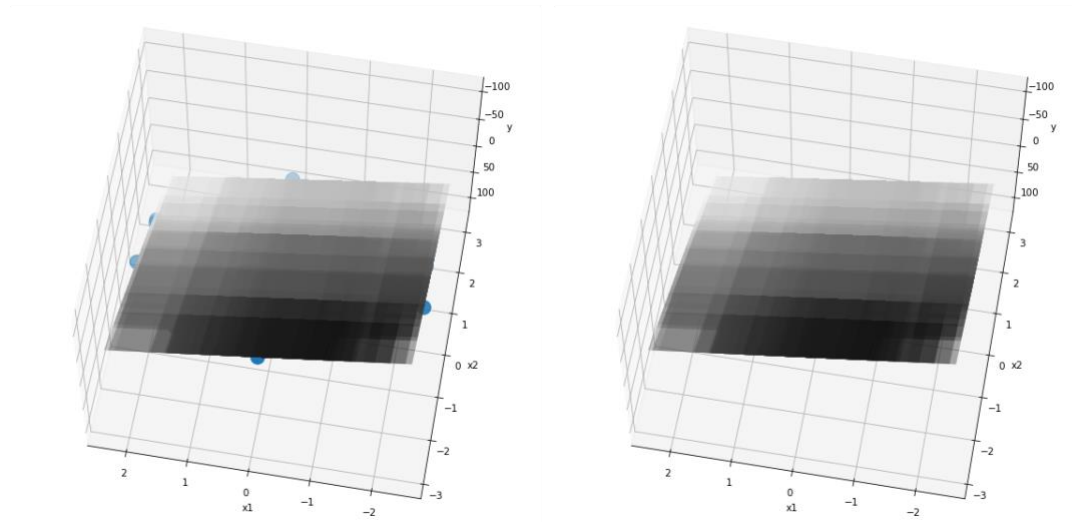


Figura 21. Rezultatele unei regresii liniare. Graficul din stanga contine predictiile facute cu acest model pentru datele de antrenament. Graficul din dreapta contine predictiile facute de acest model pentru datele de test. Observati cum regresia liniara incearca sa gaseasca planul care contine majoritatea punctelor din setul de antrenare prin reducerea erorii patratice medie dintre predictii si valorile observate.

```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

# initializeaza un set de date 2D pentru regresie
X, y = make_regression(100, 2)

# imparte datele in doua seturi (80/20) pentru antrenare si testare
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# initializeaza o regresie liniara
model = LinearRegression()

# antreneaza o regresie liniara folosind datele de test
model.fit(X_train, y_train)

# prezice valorile observatiilor pentru setul de antrenare si testare
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# calculeaza MAE (eroarea medie absoluta) si MSE (eroarea medie patratice)
mae_train = mean_absolute_error(y_true=y_train, y_pred=y_train_pred)
mae_test = mean_absolute_error(y_true=y_train, y_pred=y_train_pred)

mse_train = mean_squared_error(y_true=y_test, y_pred=y_test_pred)
mse_test = mean_squared_error(y_true=y_test, y_pred=y_test_pred)
```

```

print(f"Eroare medie absoluta pentru datele de antrenare = {mae_train}")
>> Eroare medie absoluta pentru datele de antrenare = 5.6885052224231455e-15

print(f"Eroare medie absoluta pentru datele de test = {mae_test}")
>> Eroare medie absoluta pentru datele de test = 5.6885052224231455e-15

print(f"Eroare medie patratica pentru datele de antrenare = {mse_train}")
>> Eroare medie patratica pentru datele de antrenare = 5.634439015541077e-29

print(f"Eroare medie patratica pentru datele de test = {mse_test}")
>> Eroare medie patratica pentru datele de test = 5.634439015541077e-29

# creeaza un grid 2D pentru x1 si x2 si alege 100 de observatii la intamplare
x1, x2 = np.meshgrid(X[:, 0], X[:, 1])
x1_indices = np.random.choice(range(len(x1)), 100, replace=False)
x2_indices = np.random.choice(range(len(x2)), 100, replace=False)

x1 = x1[x1_indices]
x2 = x2[x2_indices]

# extrage parametrii functiei
coefs = model.coef_
intercept = model.intercept_

# calculeaza valoarea prezisa pentru fiecare pereche (x1, x2)
zs = x1 * coefs[0] + x2 * coefs[1] + intercept

# vizualizeaza rezultate pentru setul de antrenare
_ = plt.figure(figsize=(10, 10), dpi=75)
ax = plt.axes(projection='3d')
ax.scatter3D(X_train[:, 0], X_train[:, 1], y_train, s=200)
ax.plot_surface(x1, x2, zs, alpha=0.1, rstride=1, cstride=1,
               cmap='Greys_r', edgecolor='none')
# ax.legend(['[Train] Real value', '[Train] Predicted value'])
ax.set_xlabel("x1")
ax.set_ylabel("x2")
ax.set_zlabel("y")
ax.view_init(-120, 80)
plt.show()

# vizualizeaza rezultate pentru setul de test
_ = plt.figure(figsize=(10, 10), dpi=75)
ax = plt.axes(projection='3d')
ax.scatter3D(X_test[:, 0], X_test[:, 1], y_test, s=100)
ax.plot_surface(x1, x2, zs, alpha=0.1, rstride=1, cstride=1,
               cmap='Greys_r', edgecolor='none')
# ax.legend(['[Test] Real value', '[Test] Predicted value'])
ax.set_xlabel("x1")
ax.set_ylabel("x2")
ax.set_zlabel("y")
ax.view_init(-120, 80)
plt.show()

```

Exemplul de cod 30. Implementare a regresiei liniare pentru observatii 2D in Python folosind scikit-learn si matplotlib.

## 4.5. Regresia logistică

Regresia logistică, contrar numelui, este unul dintre cei mai întâlniți algoritmi de clasificare pentru problemele în care etichetele sunt binare. Pe scurt, acest algoritm încearcă să prezică dacă o anumită observație face parte din una din cele două clase posibile. Dacă probabilitatea pentru eticheta A este mai mare de 50%, algoritmul prezice eticheta A pentru observația respectivă. În caz contrar, algoritmul prezice eticheta B (presupunând că avem doar două etichete posibile, A și B).

Regresia logistică este similară cu regresia liniară, dar în loc de o funcție liniară, încercăm să găsim o funcție logistică (sigmoidă) care ne oferă cea mai mare acuratețe. Aceasta funcție are avantajul că poate mapa valorile prezise în intervalul  $[0, 1]$ , echivalent cu o probabilitate.

De-a lungul timpului, algoritmul a fost utilizat pentru rezolvarea unor probleme diverse deoarece suportă observații cu componente discrete cât și continue. Mai mult, regresia logistică poate fi folosită și în probleme unde avem mai multe etichete prin a calcula probabilitatea apartenenței la o clasă versus cele ramase (one-vs-all).

În continuare vom vedea o implementare a regresiei logistice<sup>40</sup> folosind Python și biblioteca *scikit-learn*<sup>41</sup>. În aceste exemple încercăm să învățăm o mapare pentru un set sintetic de date generat cu funcția *make\_classification()* oferită de biblioteca *scikit-learn*. Exemplul de cod 31 reprezintă o implementare pentru o situație 1D. Acest cod se poate extinde și pentru situațiile în care observațiile au mai multe componente.

```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, roc_auc_score

# create a sample regression dataset
X, y = make_classification(100, 2, n_redundant=0, n_repeated=0)

# split the data into train and test observations
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Logistic Regression Classifier
clf = LogisticRegression()
clf.fit(X_train, y_train)

y_train_pred = clf.predict(X_train)
y_train_score = clf.predict_proba(X_train)[:, 1]

y_test_pred = clf.predict(X_test)
```

---

<sup>40</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

<sup>41</sup> <https://scikit-learn.org/stable/index.html>

```

y_test_score = clf.predict_proba(X_test)[:, 1]

# calculeaza acuratetea si "aria under the curve" pentru setul de antrenare si de test
accuracy_train = accuracy_score(y_true=y_train, y_pred=y_train_pred)
auc_train = roc_auc_score(y_true=y_train, y_score=y_train_score)

accuracy_test = accuracy_score(y_true=y_test, y_pred=y_test_pred)
auc_test = roc_auc_score(y_true=y_test, y_score=y_test_score)

print(f"Acuratetea pentru datele de antrenare = {accuracy_train}")
>> Acuratetea pentru datele de antrenare = 0.9125

print(f"Acuratetea pentru datele de test = {accuracy_test}")
>> Acuratetea pentru datele de test = 0.95

print(f"AUC pentru datele de antrenare = {auc_train}")
>> AUC pentru datele de antrenare = 0.9781113195747342

print(f"AUC pentru datele de test = {auc_test}")
>> AUC pentru datele de test = 1.0

# plot points
_, axs = plt.subplots(2, figsize=(10, 10), dpi=75)
axs[0].scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=100)
axs[0].scatter(X_train[:, 0], X_train[:, 1], c=y_train_pred, s=200, alpha=0.2)
axs[0].legend(['[Train] Real value', '[Train] Predicted value'])
axs[0].set_xlabel("x")
axs[0].set_ylabel("y")

axs[1].scatter(X_test[:, 0], X_test[:, 1], c=y_test, s=100)
axs[1].scatter(X_test[:, 0], X_test[:, 1], c=y_test_pred, s=200, alpha=0.2)
axs[1].legend(['[Test] Real value', '[Test] Predicted value'])
axs[1].set_xlabel("x")
axs[1].set_ylabel("y")

plt.show()

```

Exemplul de cod 31. Implementare a regresiei logistice pentru un exemplu 1D sintetic în Python folosind scikit-learn și matplotlib.

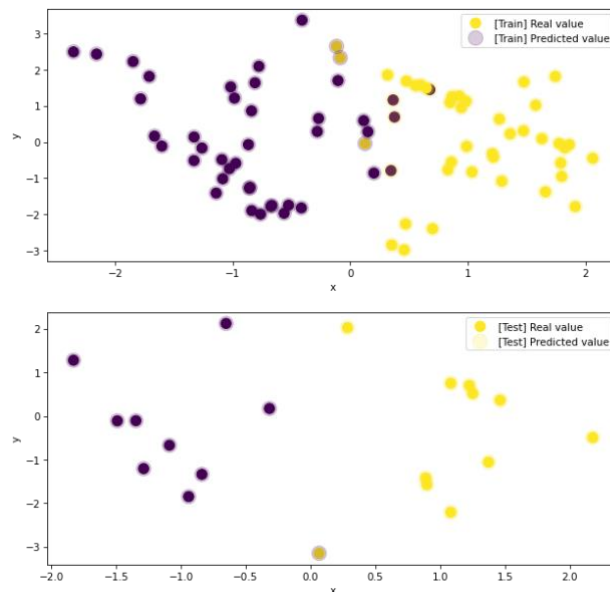


Figura 22. Rezultatele unei regresii logistice. Graficul de sus conține predicțiile făcute cu acest model pentru datele de antrenament. Graficul de jos conține predicțiile făcute de acest model pentru datele de test.

#### 4.6. Arbori de decizie pentru clasificare si regresie

În afară de regresia liniară și logistică, o altă clasă de algoritmi extrem de utilizați sunt arborii decizionali. Aceștia pot fi folosiți pentru o gamă largă de probleme, atât pentru sarcini de clasificare cât și de regresie.

Arborii decizionali sunt alcătuiți dintr-un număr variabil de noduri. Fiecare nod reprezintă o decizie bazată pe valoarea unei componente a observațiilor. Mai mult, aceasta decizie este una binară (da sau nu). Într-un arbore decizional fiecare nivel reprezintă răspunsul la una din întrebările:

- Este valoarea variabilei  $X$  mai mică decât  $X_0$ ?
- Este valoarea variabile  $X$  egală cu  $X_0$ ?

Arborii decizionali pot fi utilizați pentru variabile numerice cât și categorice deoarece nu se bazează pe o metrică continuă în luarea unei decizii. Mai degrabă, un arbore decizional se aseamănă cu un joc de „100 de întrebări” unde algoritmul pune o serie de întrebări pentru a naviga acest arbore până când ajunge la un nod final (o frunză). Acest nod final este folosit pentru predicția finală.

În exemplul din figura 23 regăsim un arbore decizional cu 2 nivele. Observați cum prima dată se verifică valoarea variabilei  $X_2$ . Dacă aceasta este mai mică decât 0.382, algoritmul verifică variabila  $X_3$ . Dacă aceasta din urmă este mai mică decât 0.661, atunci algoritmul ajunge la frunza cu predicția finală. În mod asemănător, dacă variabila  $X_2$  este mai mare decât 0.382, atunci algoritmul parcurge ramura din dreapta a arborelui până când ajunge la o frunză.

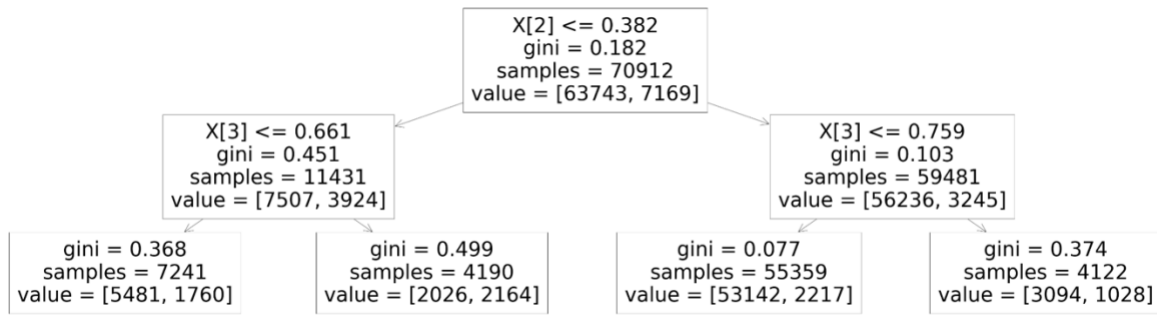


Figura 23. Un arbore decizional cu doua nivele folosit pentru o problema de clasificare. Observați că predicția finală este efectuată în baza unor decizii succesive în funcție de valorile variabilelor X2 si X3.

În cazul unei probleme de clasificare, predicția se realizează în baza unui „vot al majorității”. Cu alte cuvinte, arborele prezice eticheta cea mai comună dintre exemplele pe care le găsește în nodul final. Similar, pentru o problemă de regresie, se folosește o valoare medie a valorilor etichetelor din nodul final.

Pentru a ilustra mai bine modul în care funcționează un arbore decizional, vă propun să analizăm un exemplu simplu de clasificare. Imaginați-vă că am colectat observațiile din tabelul 2. Aceste date ne spun dacă o piață este deschisă în funcție de temperatură și prezența precipitațiilor. În baza acestui eșantion ne dorim să realizăm un arbore decizional pentru a prezice dacă piața va fi deschisă într-o altă zi.

Numărul observației	Temperatura	Precipitații	Piața deschisă?
1	10	Nu	Da
2	18.2	Nu	Da
3	-1	Nu	Nu
4	4.5	Da	Nu
5	24	Da	Da
6	4	Nu	Nu
7	13	Da	Da

Tabelul 2. Un esantion de date despre starea vremii si starea pieței (deschisa sau închisă).

Primul pas pentru a construi acest arbore decizional este să extragem pentru fiecare variabilă valorile posibile. După cum putem observa, avem doua variabile, „Temperatura” și „Precipitații”. Prima variabilă este continuă, poate avea orice valoare în intervalul [-1, 24]. Cea dea doua variabilă este discretă sau categorică cu două valori posibile, „Da” sau „Nu”. Similar, etichetele sunt la rândul lor discrete, având doar două valori, „Da” sau „Nu”. Înarmați cu aceste informații putem să începem construcția arborelui.

Vom începe cu variabila „Precipitații” și vom grupa exemplele în funcție de aceasta. În urma acestei grupări obținem primul nod din arborele nostru care răspunde la întrebarea: „Este variabila Precipitații egală cu Da?”.



Numărul observației	Temperatura (grade)	Precipitații	Piața deschisă?
4	4.5	Da	Nu
5	24	Da	Da
7	13	Da	Da
1	10	Nu	Da
2	18.2	Nu	Da
3	-1	Nu	Nu
6	4	Nu	Nu

Tabelul 3. Rezultatele grupării observațiilor în funcție de valoarea variabilei „Precipitații”.

Din Tabelul 3, observăm că folosind doar variabila „Precipitații” pentru a prezice dacă piața este deschisă folosind votul majorității pe fiecare ramură, am avea o acuratețe de 66% (în două din trei cazuri este deschisă) în cazul zilelor cu precipitații și o acuratețe de 50% (în două din patru cazuri este deschisă) în cazul zilelor fără precipitații. Rezultatele nu sunt impresionante. Dar, nu uitați, mai avem acces la niște informații pe care nu le-am folosit până acum, și anume temperatura.

Repetăm procedeul pe care l-am urmat în cazul variabilei „Precipitații”. Însă, fiind o variabilă continuă, trebuie să alegem o valoare limită a acesteia în baza căreia să luăm o decizie. Pentru a alege această valoare plecăm de la ideea că vrem să avem frunze cât mai „pure” (frunze care să conțină exemple doar dintr-o clasă). Prin urmare, putem începe cu valoarea minimă a variabilei „Precipitații”, -1, și împărțim exemplele în funcție de aceasta până când ajungem să avem frunze pure.

După câteva iterații vom ajunge la o valoare limită a temperaturii de 5 grade care împarte exemplele precum în Figura 24. Folosite împreună, cele două decizii conduc la predicții corecte 100% din cazuri. Prin urmare, am reușit să avem un model care are o acuratețe de 100% când este evaluat pe setul de antrenare. Următorul pas acum ar fi să încercăm acest arbore decizional pentru situații pe care nu le-am mai întâlnit. În acest fel putem să avem o evaluare obiectivă a performanței modelului nostru.

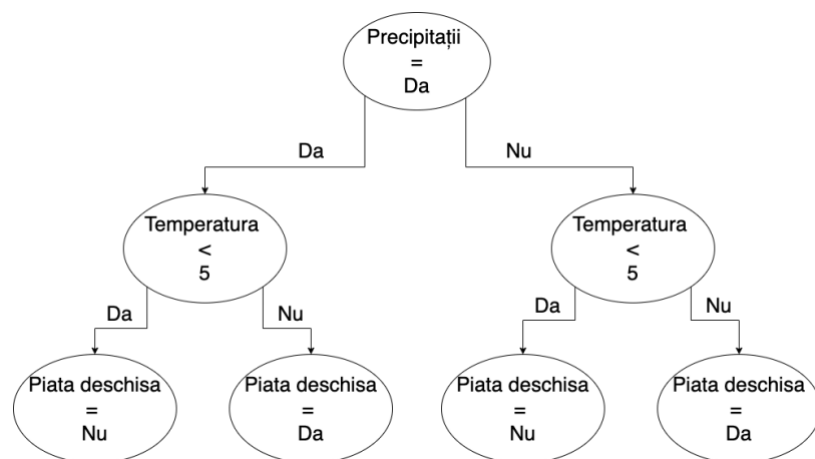


Figura 24. Arbore decizional pentru a prezice când piața este deschisă în funcție de temperatura și prezența precipitațiilor.

```

import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.preprocessing import StandardScaler

from toolbox import *

# load data
df = pd.read_csv("weatherHistory.csv")

# clean data
df = remove_df_outliers_iqr(df)
X, y = split_data(df)

column = "Wind Bearing (degrees)"
x_component = np.cos(X[column])
y_component = np.sin(X[column])

X["Wind Bearing (Ox)"] = x_component
X["Wind Bearing (Oy)"] = y_component
X = X.drop(["Wind Bearing (degrees)"], axis=1)

# remove correlated features
corr_matrix = compute_correlation_matrix(X)
X = remove_correlated_columns(corr_matrix, X)
X.drop(["Temperature (C)"], inplace=True, axis=1)

# encode target variable
y[y == "rain"] = 0
y[y == "snow"] = 1

# split data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
y_train = np.array(y_train, dtype=float)
y_test = np.array(y_test, dtype=float)

# scale features so that they have mean 0 and std 1
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Decision Trees (classification)
clf = DecisionTreeClassifier(max_depth=3)
clf = clf.fit(X_train, y_train)

fig = plt.figure(figsize=(75, 20))
plot_tree(clf, ax=fig.gca(), feature_names=X.columns)
plt.show()

y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)

```

```
train_acc = accuracy_score(np.array(y_train, dtype=float), y_train_pred)
test_acc = accuracy_score(np.array(y_test, dtype=float), y_test_pred)

print(f"Training set accuracy: {train_acc * 100} %. Test set accuracy: {test_acc * 100} %.")
>> Training set accuracy: 90.43039259927798 %. Test set accuracy: 90.09532404534943 %.
```

Exemplul de cod 32. Un arbore decizional pentru predicția precipitațiilor (ploaie sau zăpadă).

În acest exemplu puteți găsi câteva transformări adiționale ale datelor. În prima parte a codului sunt eliminate valorile greșite și componentele corelate. Ulterior, variabilele rămase sunt transformate astfel încât fiecare coloană să aibă media 0 și deviația standard 1.

În practică, există mai mulți algoritmi pentru căutarea valorilor limită folosite pentru a lua o decizie la nivel de nod. Ideea principală a acestora fiind similară cu cea din exemplul de mai sus. Dacă doriți să aflați mai multe despre modalitățile de antrenare a arborilor de decizie și clasificare, puteți să consultați bibliografia sau referința de mai jos<sup>42</sup>. De asemenea, în exemplul de cod 32 vom antrena un arbore decizional pentru a clasifica tipul precipitațiilor (ploaie sau zăpadă) folosind observații despre diferiți parametri ai vremii precum viteza vântului, vizibilitate, sau presiunea atmosferică.

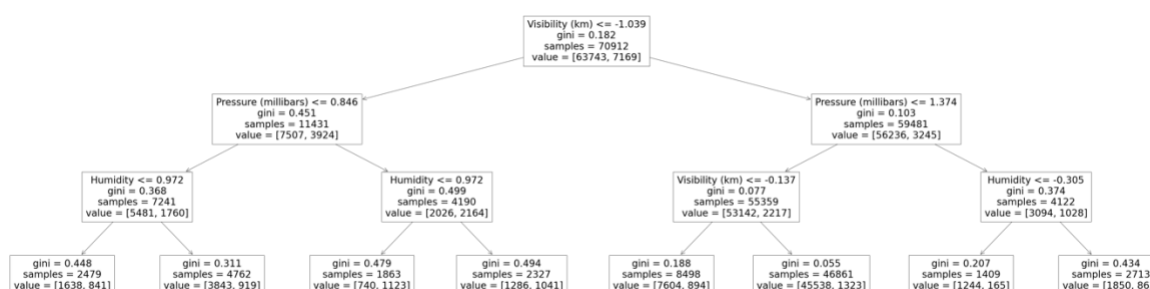


Figura 25. Arborele decizional antrenat folosind exemplul de cod 33.

Arborii decizionali sunt foarte eficienți pentru date tabelare, uneori depășind chiar performanța unor algoritmi mult mai complecși. Din păcate însă, în varianta lor de bază, fără nici un fel de optimizări, aceștia au tendința de a se supra adapta la setul de antrenare. În practică, aducem unele modificări acestor arbori. De exemplu, limităm adâncimea (cate noduri sunt folosite pentru a face o predicție) sau antrenăm mai mulți arbori cu o adâncime mică (evident cu o performanță mai slabă) și combinăm predicțiile tuturor pentru a face o predicție finală.

## 4.7. Ansambluri

În secțiunea 4.2. am văzut cum algoritmi de învățare pot manifesta bias sau varianță. În cele ce urmează vom discuta despre o metodă prin care putem să îmbunătățim performanța algoritmilor și să reducem varianța acestora.

<sup>42</sup> <http://scikit-learn.org/stable/modules/tree.html>

Ansamblurile constituie o metodă de învățare prin care folosim mai mulți algoritmi (de obicei cu o complexitate redusă) pentru a obține o performanță mai bună decât am fi putut obține folosind oricare din algoritmi considerați. Un ansamblu este mulțime finită de modele alternative.

Există situații în care nu putem selecta modelul (maparea) potrivit pentru problema pe care dorim să o rezolvăm (chiar dacă acesta există). Din acest motiv putem considera o suită de modele mai „slabe” pe care să le combinăm pentru a obține performanța dorită. De obicei această metodă dă rezultate bune dacă includem în ansamblu modele diferite (tipuri de modele diferite). Cu cât diversitatea modelelor este mai mare, cu atât avem șanse să obținem rezultate mai bune. Cu alte cuvinte, dacă avem multe modele „slabe”, care nu au o performanță prea bună, pe care le combinăm într-un ansamblu vor ajunge să producă rezultate mult mai precise. Presupunând că rezolvăm o problemă de regresie și avem un ansamblu de modele  $\{M_1, M_2, M_3\}$ , cea mai simplă variantă prin care să le creștem puterea de predicție ar fi să folosim media predicțiilor individuale. Cu alte cuvinte, facem o predicție cu fiecare model și ulterior calculăm media acestora astfel:

$$\begin{aligned}\widehat{y}_1 &= M_1(\vec{x}) \\ \widehat{y}_2 &= M_2(\vec{x}) \\ \widehat{y}_3 &= M_3(\vec{x}) \\ \widehat{y} &= \frac{\widehat{y}_1 + \widehat{y}_2 + \widehat{y}_3}{3}\end{aligned}$$

Mai mult, presupunând acum că modelul  $M_1$  are o performanță mai mare decât celelalte modele, putem extinde acest concept al predicției medii prin a oferi predicției modelului  $M_1$  o pondere mai mare în calculul predicției medii astfel:

$$\begin{aligned}\widehat{y}_1 &= M_1(\vec{x}) \\ \widehat{y}_2 &= M_2(\vec{x}) \\ \widehat{y}_3 &= M_3(\vec{x}) \\ \widehat{y} &= w_1\widehat{y}_1 + w_2\widehat{y}_2 + w_3\widehat{y}_3, \\ &\text{unde } w_1 + w_2 + w_3 = 1 \\ &\text{și } w_1 \geq w_2, w_1 \geq w_3\end{aligned}$$

Mai jos veți găsi două exemple de ansambluri folosite pentru o problemă de regresie (primul care folosește predicția medie și cel de-al doilea care folosește media ponderată). După cum putem observa, predicția făcută de ansamblu este mult mai bună decât predicțiile individuale ale modelelor.

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
```

```

from sklearn.metrics import mean_absolute_error, mean_squared_error

# create a sample regression dataset
X, y = make_regression(10000, 10)

# split the data into train and test observations
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# create and train two models
# K Nearest Neighbours Regressor
knn_regressor = KNeighborsRegressor(1)
knn_regressor.fit(X_train, y_train)

# Support Vector Regressor
svr_regressor = SVR(kernel="poly", degree=5)
svr_regressor.fit(X_train, y_train)

# make predictions for each model
y_pred1 = knn_regressor.predict(X_test)
y_pred2 = svr_regressor.predict(X_test)

# compute average prediction
avg_pred = np.mean([y_pred1, y_pred2], axis=0)

# report error for each model and the ensemble
print("Ensembler MAE:", mean_absolute_error(y_test, avg_pred))
>> Ensembler MAE: 72.0162594402013

print("KNN MAE:", mean_absolute_error(y_test, y_pred1))
>> KNN MAE: 72.02540000539632

print("SVM MAE:", mean_absolute_error(y_test, y_pred2))
>> SVM MAE: 101.03643270798956

print("Ensembler MSE:", mean_squared_error(y_test, avg_pred))
>> Ensembler MSE: 8120.550950501643

print("KNN MSE:", mean_squared_error(y_test, y_pred1))
>> KNN MSE: 8214.29737613626

print("SVR MSE:", mean_squared_error(y_test, y_pred2))
>> SVR MSE: 16350.828512246919

```

Exemplul de cod 33. Un ansamblu de două modele (kNN si SVM) pentru o problemă de regresie. Observați cum ansamblul are performanța mai bună (atât MAE cat si MSE sunt mai mici) comparativ cu fiecare dintre cele două modele separate.

```

from sklearn.neighbors import KNeighborsRegressor

```

```

from sklearn.svm import SVR
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

# create a sample regression dataset
X, y = make_regression(10000, 10)

# split the data into train and test observations
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# create and train two models
# K Nearest Neighbours Regressor
knn_regressor = KNeighborsRegressor(1)
knn_regressor.fit(X_train, y_train)

# Support Vector Regressor
svr_regressor = SVR(kernel="poly", degree=5)
svr_regressor.fit(X_train, y_train)

# make predictions for each model
y_pred1 = knn_regressor.predict(X_test)
y_pred2 = svr_regressor.predict(X_test)

# compute weighted average prediction
w1 = 0.6
w2 = 0.4
weighted_pred = (w1 * y_pred1 + w2 * y_pred2)

print("Ensembler MAE:", mean_absolute_error(y_test, weighted_pred))
>> Ensembler MAE: 69.0653635230316

print("KNN MAE:", mean_absolute_error(y_test, y_pred1))
>> KNN MAE: 75.45120081481208

print("SVR MAE:", mean_absolute_error(y_test, y_pred2))
>> SVR MAE: 100.35660195357568

print("Ensembler MSE:", mean_squared_error(y_test, avg_pred))
>> Ensembler MSE: 50733.565991515774

print("KNN MSE:", mean_squared_error(y_test, y_pred1))
>> KNN MSE: 8819.73021963769

print("SVR MSE:", mean_squared_error(y_test, y_pred2))
>> SVR MSE: 16757.74755999095

```

Exemplul de cod 34. Un ansamblu de două modele (kNN si SVM) pentru o problemă de regresie. Predicția ansamblului alocă o pondere mai mare predicției făcute de kNN decât celei

facute de SVM. Observați cum ansamblul are performanța mai bună (atât MAE cat si MSE sunt mai mici) comparativ cu fiecare dintre cele două modele separate.

Ideea unui ansamblu poate fi folosită și în cazul problemelor de clasificare, cu o mică modificare. În loc sa raportăm media predicțiilor, vom considera că fiecare model este un participant într-un forum unde își poate spune părerea (ce etichetă crede că are o anumită observație). După ce fiecare clasificator votează, vom alege eticheta care a adunat cele mai multe voturi. În exemplul de mai jos aplicăm aceasta metodă pentru a prezice specia unui Iris.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import VotingClassifier

# create a sample regression dataset
X, y = make_classification(1000, 10)
# split the data into train and test observations
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
#create and train three models
# K Nearest Neighbours Classifier
knn = KNeighborsClassifier(3)
knn.fit(X_train, y_train)

# Support Vector Classifier
svm = SVC(kernel="poly", degree=5)
svm.fit(X_train, y_train)

# Logistic Regression Classifier
lrc = LogisticRegression()
lrc.fit(X_train, y_train)

estimators = [('KNN', knn), ('Logistic Regression', lrc), ('SVC', svm)]
maxVotingClassifier = VotingClassifier(estimators=estimators, voting='hard')
maxVotingClassifier.fit(X_train, y_train)

y_pred = maxVotingClassifier.predict(X_test)
print("Max Voting Classifier Accuracy:" accuracy_score(y_test, y_pred)*100)
>> Max voting classifier accuracy: 86.5
```

Exemplul de cod 34. Un ansamblu de două modele (kNN si SVM) pentru o problemă de clasificare.

Metodele prezentate nu sunt singurele variante prin care putem sa balansam biasul si varianta unui model. Pentru mai multe detalii despre metode mai avansate precum bagging sau boosting, consultați referințele (Bishop, 2006).

## Exerciții

1. Implementați un algoritm de regresie logistica pentru setul de date *load\_breast\_cancer()*<sup>43</sup> din *scikit-learn*. Studiați comportamentul algoritmului pentru mai multe valori ale parametrilor acestuia.
2. Implementați un algoritm de regresie liniară pentru setul de date *load\_boston()*<sup>44</sup> din *scikit-learn*. Studiați comportamentul algoritmului pentru mai multe valori ale parametrilor acestuia.
3. Implementați un arbore decizional pentru setul de date *load\_breast\_cancer()*<sup>45</sup> din *scikit-learn*. Studiați comportamentul algoritmului pentru mai multe valori ale parametrilor acestuia.
4. Studiați algoritmul *Radom Forest* cu ajutorul cărora reducem varianța arborilor decizionali și aplicați-l pe setul de date *load\_breast\_cancer()*<sup>46</sup> din *scikit-learn*. Studiați comportamentul algoritmului pentru mai multe valori ale parametrilor acestuia.
5. Studiați diferite metode de regularizare în contextul regresiei liniare și comparați rezultatele pentru problema predicțiilor prețurilor caselor din setul de date *load\_boston()*<sup>47</sup> din *scikit-learn*.

## Bibliografie

- Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC. ISBN 0387310738.
- Botchkarev, A. (2019). A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms. *Interdisciplinary Journal of Information, Knowledge, and Management*, 045-076. Preluat de pe arxiv: <https://arxiv.org/abs/1809.03006>
- Czakon, J. (2022, 01 05). *Neptune.ai*. Preluat de pe Neptune blog: <https://neptune.ai/blog/evaluation-metrics-binary-classification>
- Deisenroth, M., Faisal, A., & Ong, C. (2020). *Mathematics for Machine Learning*. Cambridge University Press. ISBN 9781108455145.

---

<sup>43</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_breast\\_cancer.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html)

<sup>44</sup> [https://scikit-learn.org/0.24/modules/generated/sklearn.datasets.load\\_boston.html#sklearn.datasets.load\\_boston](https://scikit-learn.org/0.24/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston)

<sup>45</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_breast\\_cancer.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html)

<sup>46</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_breast\\_cancer.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html)

<sup>47</sup> [https://scikit-learn.org/0.24/modules/generated/sklearn.datasets.load\\_boston.html#sklearn.datasets.load\\_boston](https://scikit-learn.org/0.24/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston)



- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. ISBN 9780262035613.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer Science+Business Media. ISBN 9781461471370.
- Jang, J.-S. R., Sun, C. T., & Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*. Upper Saddle River: Prentice Hall, Inc. ISBN 9780132610667.
- Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press. ISBN 9780262018029.
- Negnevitsky, M. (2005). *Artificial Intelligence - A guide to intelligent systems*. Pearson Education Limited. ISBN 0321204662.
- Rothman, D. (2020). *Hands-On Explainable AI (XAI) with Python: Interpret, Visualise, Explain, and Integrate Reliable AI for Fair, Secure, and Trustworthy AI Apps*. Packt Publishing. ISBN 9781800208131.
- Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press. ISBN 9781107057135.
- Sivanandam, S. N., & Deepa, S. (2011). *Principles of Soft Computing*. New Delhi: Wiley India Pvt Ltd. ISBN 9788126527410.

## 5. Învățarea nesupravegheata

### 5.1. Noțiuni fundamentale

În secțiunea anterioară am văzut cum putem să învățăm regulile de funcționare ale unui sistem, proces sau fenomen plecând de la serie de observații etichetate despre acesta. În practică nu avem tot timpul luxul de a avea observații etichetate. Prin urmare, avem nevoie de o abordare nouă prin intermediul căreia să putem oferi unei mașini abilitatea de a identifica cel puțin structura observațiilor. Rezultatul acestui exercițiu poate fi folosit ca o etichetare inițială pe care ulterior să o corectăm manual în vederea dezvoltării de algoritmi de învățare supravegheată. Aceste tehnici poartă numele de învățare nesupravegheată. Pentru a înțelege ce putem face cu ajutorul învățării nesupravegheate cred că cel mai simplu ar fi să analizăm o problemă.

Imaginați-vă că avem un site de e-commerce. Acest site este vizitat zilnic de un număr variabil de cumpărători, o parte dintre aceștia recurenți. Mai mult, avem și un sistem informatic cu ajutorul căruia colectăm tot ce se întâmplă pe site-ul nostru. Din păcate, legislația în vigoare nu ne permite să înregistrăm nici un fel de date personale ale utilizatorilor. Deoarece ne dorim să ne creștem vânzările, realizăm o campanie de publicitate online. Pentru a realiza această campanie cât mai eficient, vrem să generăm anunțuri personalizate pentru diferite grupe de clienți, care au comportamente sau preferințe similare. Cum putem face acest lucru?

Primul pas este să analizăm problema dintr-o altă perspectivă. Avem un set de date  $\{(\vec{x}_i)\}, i = \{1, 2, 3, \dots, m\}$  care reprezintă toate detaliile colectate de sistemul nostru. Acestea pot fi marcajul temporal al vizitei, pagina vizitată, timpul petrecut pe o pagină, click-urile pe care le-a dat utilizatorul, și alte detalii relevante. Pentru a simplifica analiza, ne vom imagina ca avem doar două informații, pagina vizitată și timpul petrecut pe acea pagină.

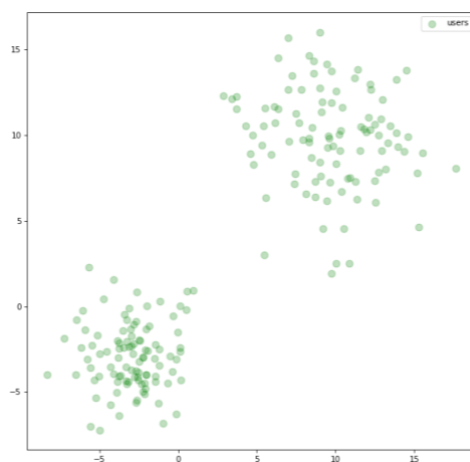


Figura 26. Diagrama de dispersie ce conține informații despre utilizatorii site-ului. Fiecare punct reprezintă activitatea  $(x_1, x_2)$  unui utilizator. Aceștia pot fi utilizatori recurenți ca și utilizatori unici. Datele au o structură destul de evidentă, utilizatorii formând două grupuri distincte.

Orice problemă devine mai ușor de rezolvat atunci când putem să o reprezentăm într-o formă grafică. Prin urmare, vom începe prin a reprezenta informațiile pe care le avem despre cumpărători ca o pereche  $(x_1, x_2)$  pe care o vizualizăm folosind o diagramă de dispersie (Figura 26).

Pentru a putea personaliza campaniile publicitare avem nevoie de un criteriu prin care să putem împărți automat acești utilizatori în grupe distincte, în funcție de valorile  $(x_1, x_2)$ . Dacă reușim să găsim acest criteriu, putem ulterior să investigăm care sunt caracteristicile utilizatorilor din fiecare grupă sau putem chiar să îi atribuim pe aceștia în mod automat într-o grupă. Activitatea lor poate fi astfel folosită pentru a le personaliza experiența pe pagina noastră.

Tot din figura 26 observăm ca utilizatorii sunt grupați în funcție de niște asemănări dintre valorile perechilor  $(x_1, x_2)$ . Cu alte cuvinte, utilizatori pentru care  $(x_1, x_2)$  sunt similare vor fi în aceeași regiune a spațiului generat de cele două componente ale observațiilor.

Dar ce înseamnă pentru o mașină ca perechile sunt similare? În exemplul nostru, observăm că fiecare caracteristică a unui utilizator definește o axă din spațiul 2D unde care i-am reprezentat. Putem folosi aceasta idee să propunem următoarea definiție pentru două perechi similare.

Două perechi  $(x_1, x_2)_i$  și  $(x_1, x_2)_j$  sunt similare dacă depărtarea dintre acestea este mai mică față de depărtarea de o a treia pereche  $(x_1, x_2)_k$ .

Putem înțelege această definiție dacă ne concentrăm, cel puțin pentru moment, doar pe una din componentele unei perechi. Cu această simplificare ajungem la următoarea situație.

1. Avem trei observații diferite pentru o componentă, să presupunem  $x_1$ . În Figura 27 am reprezentat această situație generică.
2. Definim cât de apropiate sunt aceste observații ca  $d_{ij} = (x_{i1} - x_{j1})^2$ . Această definiție a distanței se poate extinde și pentru mai multe dimensiuni așa cum am văzut atunci când am introdus algoritmul k Nearest Neighbors.
3. Etichetăm două observații ca fiind similare dacă  $d_{ij} < d_{ik}$  și  $d_{ij} < d_{jk}$ , unde  $i, j, k$  denotă cele trei observații.
4. În cazul în care avem egalitate, folosim o regulă cu ajutorul căreia luăm o decizie. De exemplu, putem alege la întâmplare.

Mai mult, dacă extindem aceeași abordare pentru ambele componente ale unei perechi și redefinim distanța dintre observații ca  $d_{ij} = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2$  vom obține situația reprezentată în figura 28.

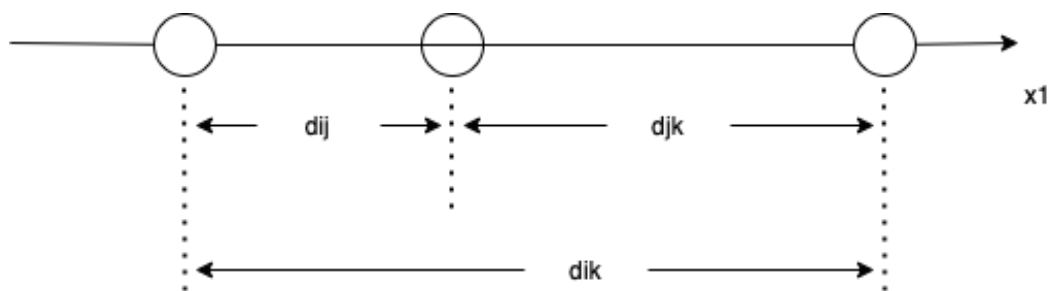


Figura 27. Exemplu a trei observații diferite și a distanțelor dintre acestea în 1D. Deoarece distanța dintre primele două observații din stânga este mai mică decât toate celelalte distanțe, am putea concluziona că acestea sunt similare.

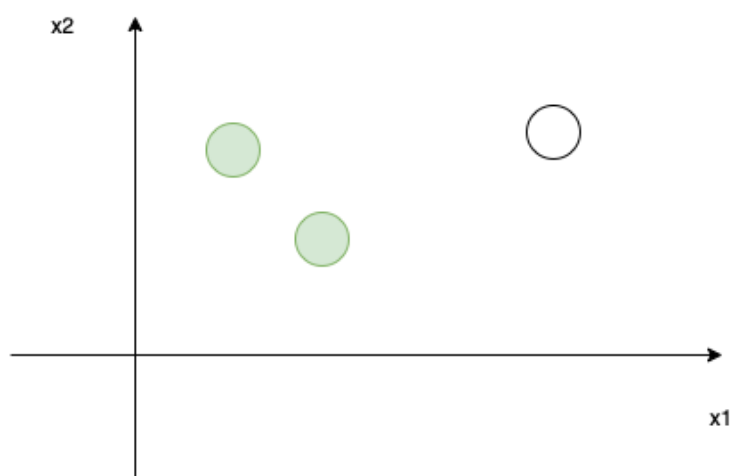


Figura 28. Exemplu a trei observații diferite și a distanțelor dintre acestea în 2D. Deoarece distanța dintre primele două observații din stânga este mai mică decât toate celelalte distanțe, am putea concluziona că acestea sunt similare.

Mai mult, am putea decide asupra unui prag superior al distanței dintre două observații. Această limită reprezintă raza unei sfere multidimensionale în care ne așteptăm să se regăsească toate punctele similare.

Exemplul prezentat aici nu este singura variantă de învățare nesupravegheată. Am ales să ne concentrăm pe acesta deoarece este cel mai des întâlnit în practică. În realitate avem mult mai multe situații în care ne dorim să înțelegem nu doar cum sunt grupate observațiile, dar și care e structura acestor date, de exemplu, dacă acestea aparțin unor anumite distribuții statistice.

Plecând de la ideile de mai sus, putem formula diferiți algoritmi prin care să automatizăm gruparea astfel încât să căutam un număr predefinit de grupuri sau să descoperim toate grupurile posibile. În cele ce urmează vom vedea unul dintre acești algoritmi de grupare, și anume algoritmul K-Means.

## 5.2. K-Means

K-Means este unul dintre cele mai des întâlniți algoritmi în învățarea nesupravegheată și probabil unul dintre cei mai intuitivi algoritmi. Obiectivul acestuia este să găsească grupuri într-un eșantion de observații fără etichete. Pentru a realiza acest lucru algoritmul are nevoie de un singur parametru, numărul de grupuri pe care le căutăm.

K-Means se bazează pe distanța dintre puncte. Mai mult, K-Means definește grupurile în funcție de locația unui punct central, numit centroid, sau centrul de „greutate” al punctelor. Obiectivul algoritmului este să găsească locația unui număr predefinit de centre pentru care grupurile conțin observații similare. În același timp, ne dorim ca aceste grupuri să fie cât mai depărtate unele de altele în raport cu distanța dintre centrele acestora. În acest fel ne „asigurăm” că nu introducem puncte care nu sunt similare într-un anumit grup.

Algoritmul urmează următorii pași:

1. Inițializăm centrele grupurilor,  $\{c_1, \dots, c_k\}$ , la întâmplare.
2. Repetăm pașii de mai jos până când poziția centrelor nu se mai modifică
  - a. Actualizăm eticheta punctelor prin a seta pentru fiecare punct  $i$  o nouă valoare a etichetei conform ecuației de mai jos:

$$K_i = \operatorname{argmin}_j \|x_i - c_j\| \quad [\text{E22}]$$

- b. Actualizăm poziția centrelor prin a seta pentru fiecare grup un nou centru conform ecuației de mai jos:

$$c_i = \frac{\sum_1^m 1_{\{k_i=j\}} x_i}{\sum_1^m 1_{\{k_i=j\}}} \quad [\text{E23}]$$

Unde  $K_i$  reprezintă grupul care îi este atribuit observației  $x_i$

În continuare vom vedea un exemplu în Python unde implementăm K-Means pentru  $k = 2$ , folosind biblioteca *scikit-learn*<sup>48</sup>. De asemenea, în figura 29 putem vedea rezultatele acestui algoritm unde observațiile au fost împărțite automat în două grupuri (un grup etichetat cu mov și un grup etichetat cu galben).

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
```

```
X, y = make_classification(100, 2, n_redundant=0, n_repeated=0)
```

```
# find two clusters
```

---

<sup>48</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

```

kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

clusters = kmeans.predict(X)

_ = plt.figure(figsize=(10, 10), dpi=75)
plt.scatter(X[:,0], X[:, 1], c=clusters, s=100)
plt.savefig('kmeans-1.png')
plt.show()

# find four clusters
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)

clusters = kmeans.predict(X)

_ = plt.figure(figsize=(10, 10), dpi=75)
plt.scatter(X[:,0], X[:, 1], c=clusters, s=100)
plt.savefig('kmeans-2.png')
plt.show()

```

Exemplul de cod 35. Implementarea algoritmului KMeans folosind scikit-learn.

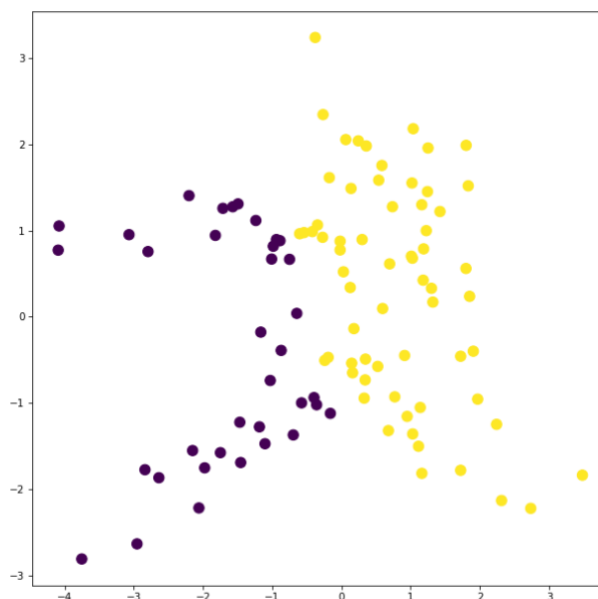


Figura 29. Rezultatele grupării folosind K-Means cu  $k = 2$ .

Valoarea parametrului  $k$  influențează modul în care algoritmul împarte observațiile în mai multe grupuri. În funcție de valoarea acestui parametru avem rezultate diferite. De exemplu, în figura 30, am modificat valoarea lui  $k$  de la  $k = 2$  la  $k = 4$ . Observați noua grupare a punctelor.

Spre deosebire de învățarea supravegheată, în cazul învățării nesupravegheate nu avem nici o metodă obiectivă cu ajutorul căreia să stabilim dacă grupările pe care le facem sunt reale. Neavând etichete este imposibil să știm cum ar trebui împărțite punctele. Prin urmare, în

învățarea automată ne bazăm pe alte metode pentru a stabili sau pentru a compara performanța unor grupări sau algoritmi.

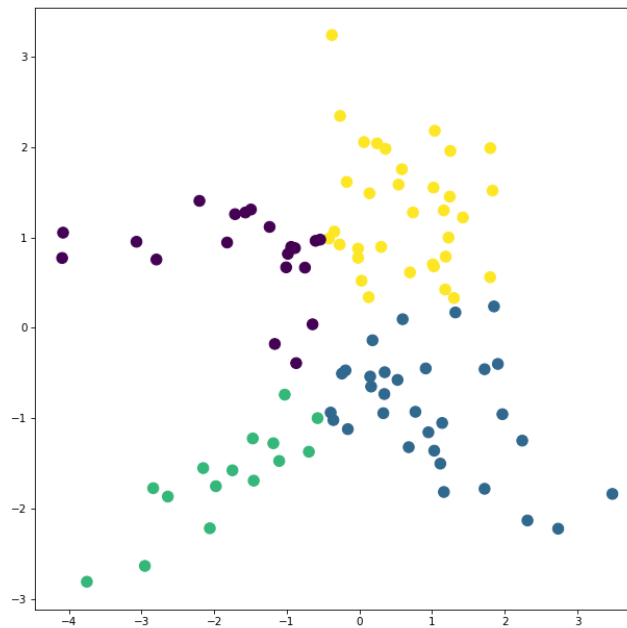


Figura 30. Rezultatele grupării folosind K-Means cu  $k = 4$ .

Prima posibilitate pentru a stabili validitatea rezultatelor unei grupări este să consultăm un expert în domeniul pentru care am realizat gruparea. După ce ajungem la un rezultat, putem oferi acestor experți o serie de exemple și etichetele generate pentru ca aceștia să evalueze dacă acestea sunt valide. O altă variantă este să prezentăm acestor experți o suită de grafice precum cele din figurile 29 și 30 împreună cu o modalitate prin care aceștia să poată interoga valorile fiecărui punct în vederea stabilirii corectitudinii grupării. În cazul în care gruparea nu este validată, este necesar să revenim asupra ei, încercând un număr diferit de grupuri sau poate alte metode. Acest proces poate necesita foarte mult timp și este predispus greșelilor umane. Este posibil că grupările descoperite de algoritmul nostru să nu se ridice la așteptările experților. Acest ultim punct este foarte important. Faptul că experți nu consideră o grupare ca fiind potrivită nu înseamnă că acea grupare este greșită ci doar diferită. Prin urmare este nevoie de o analiză mai detaliată a rezultatelor pentru a ajunge la concluzie.

Putem evalua performanța unei grupări plecând de la două noțiuni intuitive, suma distanțelor pătratice în interiorul unui grup, [E24], și suma distanțelor pătratice dintre grupări, [E25]. În baza acestor două metrici putem construi diferiți coeficienți care ne arată cât de eficientă este o grupare în a eticheta punctele similare.

$$SSW = \sum_{i=1}^m \|\mathbf{x}_i - \mathbf{c}_{k_i}\|^2 \quad [\text{E24}]$$

Unde  $k_i$  este identificatorul grupului din care face parte observația  $i$  și  $m$  este numărul de observații. Acesta metrică ne oferă informații despre cât de diferită este o observație față de media observațiilor. O valoare mică a acestei metrici înseamnă o grupare în care elementele nu sunt foarte diferite.

$$SSB = \sum_{i=1}^k n_i \|\bar{x} - c_i\|^2 \text{ [E25]}$$

Unde  $n_i$  reprezintă numărul de elemente din grupul  $i$ ,  $\bar{x}$  este poziția medie a tuturor grupelor și  $c_i$  este poziția unei grupări. Această metrică ne oferă informații despre cât de apropiate sunt grupurile descoperite. O valoare mare a acestei metrici înseamnă că am realizat o separare foarte bună a elementelor diferite.

Folosind aceste metrici, putem defini alte metrici cu ajutorul cărora să evaluăm performanța algoritmului. În cele ce urmează vom introduce câteva dintre aceste metrici fără să petrecem prea mult timp explicându-le. Toate metricile de mai jos se bazează pe ideea că o observație aparține doar unei singure grupări. Cu alte cuvinte, dacă o observație aparține grupului  $c_i$  aceasta nu poate să aparțină și altui grup. Mai mult, dacă un grup este identificat corect, ne așteptăm că toate observațiile din interiorul acestuia să fie apropiate și distanța dintre grupuri să fie cât mai mare. Cei interesați pot consulta Evritt, Landau, Leese & Stahl (2011) pentru mai multe detalii sau notele de subsol aferente fiecărei metrici.

#### 1. Calinski – Harbusz Index<sup>49</sup>

$$CH = \frac{\frac{SSB}{k-1}}{\frac{SSW}{m-k}} \text{ [E26]}$$

#### 2. Hartingan Index<sup>5051</sup>

$$H = \left( \frac{SSW_k}{SSW_{k+1}} - 1 \right) (m - k - 1) \text{ [E27]}$$

#### 3. Dunn's Index<sup>5253</sup>

$$D = \frac{\min_{i=1,M} \min_{j=i+1,M} d(c_i, c_j)}{\max_{k=1,M} diam(c_k)} \text{ [E28]},$$

Unde,

$$d(c_i, c_j) = \min_{x \in c_i, x' \in c_j} \|x - x'\|^2 \text{ [E29]}$$

<sup>49</sup> <https://pyshark.com/calinski-harabasz-index-for-k-means-clustering-evaluation-using-python/>

<sup>50</sup> [https://www.bx.psu.edu/old/courses/bx-fall04/How\\_Many\\_Clusters.pdf](https://www.bx.psu.edu/old/courses/bx-fall04/How_Many_Clusters.pdf)

<sup>51</sup> [https://epub.ub.uni-muenchen.de/12797/1/DA\\_Scherl.pdf](https://epub.ub.uni-muenchen.de/12797/1/DA_Scherl.pdf)

<sup>52</sup> <https://mayankdw.medium.com/k-means-clustering-and-dunn-index-implemantaion-from-scratch-9c66573bfe90>

<sup>53</sup> <https://pyshark.com/dunn-index-for-k-means-clustering-evaluation-using-python/>



$$\text{diam}(c_k) = \max_{x \in c_k, x' \in c_k} \|x - x'\|^2 \text{ [E30]}$$

În concluzie, învățarea nesupravegheată ne oferă o colecție de algoritmi cu care putem identifica structurile matematice prezente în observațiile noastre atunci când nu avem acces direct la etichetele fiecărei observații. Este folosit pentru a identifica grupuri de puncte de date similare și pentru a înțelege structura de bază a datelor. Cele mai frecvent utilizate tehnici de învățare nesupravegheată sunt algoritmii de clustering, cum ar fi K-Means.

În plus, alte tehnici de învățare nesupravegheată, cum ar fi analiza componentelor principale, pot fi utilizate pentru a reduce dimensionalitatea datelor. Aceste tehnici depășesc scopul acestei lucrări, dar dacă sunteți interesați puteți consulta bibliografia de la sfârșitul acestei secțiuni pentru a afla mai multe.

Învățarea nesupravegheată poate fi utilizată într-o varietate de aplicații, de la segmentarea clienților până la detectarea anomaliilor și multe altele. În general, învățarea nesupravegheată este un instrument puternic care ne oferă capacitatea de a descoperi modele și structuri în date fără cunoștințe sau etichete anterioare.

## Exerciții

1. Implementați algoritmul K-Means fără să folosiți biblioteca scikit-learn. În implementarea voastră puteți folosi NumPy. Puteți folosi setul de date *load\_iris()*<sup>54</sup> din *scikit-learn*.
2. Implementați algoritmul K-Means pentru a crea o grupare a datelor din setul *load\_iris()*<sup>55</sup> pentru diferite valori ale lui *k*.
3. Implementați algoritmul K-Means folosind datele din setul *load\_iris()*<sup>56</sup> și determinați valoarea cea mai potrivită pentru *k* folosind metoda „Elbow method”<sup>57</sup>.
4. Implementați indicii 1 – 3 pentru a măsura performanța unui algoritm precum K-Means pe setul de date *load\_iris()*<sup>58</sup>.

<sup>54</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_iris.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)

<sup>55</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_iris.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)

<sup>56</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_iris.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)

<sup>57</sup> <https://www.oreilly.com/library/view/statistics-for-machine/9781788295758/c71ea970-0f3c-4973-8d3a-b09a7a6553c1.xhtml>

<sup>58</sup> [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_iris.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)

## Bibliografie

- Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC. ISBN 0387310738.
- Deisenroth, M., Faisal, A., & Ong, C. (2020). *Mathematics for Machine Learning*. Cambridge University Press. ISBN 9781108455145.
- Evritt, B. S., Landau, S., Leese M., & Stahl, D. (2011). *Cluster analysis*. John Wiley & Sons, Ltd. ISBN 9780470749913.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer Science+Business Media. ISBN 9781461471370.
- Jang, J.-S. R., Sun, C. T., & Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*. Upper Saddle River: Prentice Hall, Inc. ISBN 9780132610667.
- Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press. ISBN 9780262018029.
- Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press. ISBN 9781107057135.