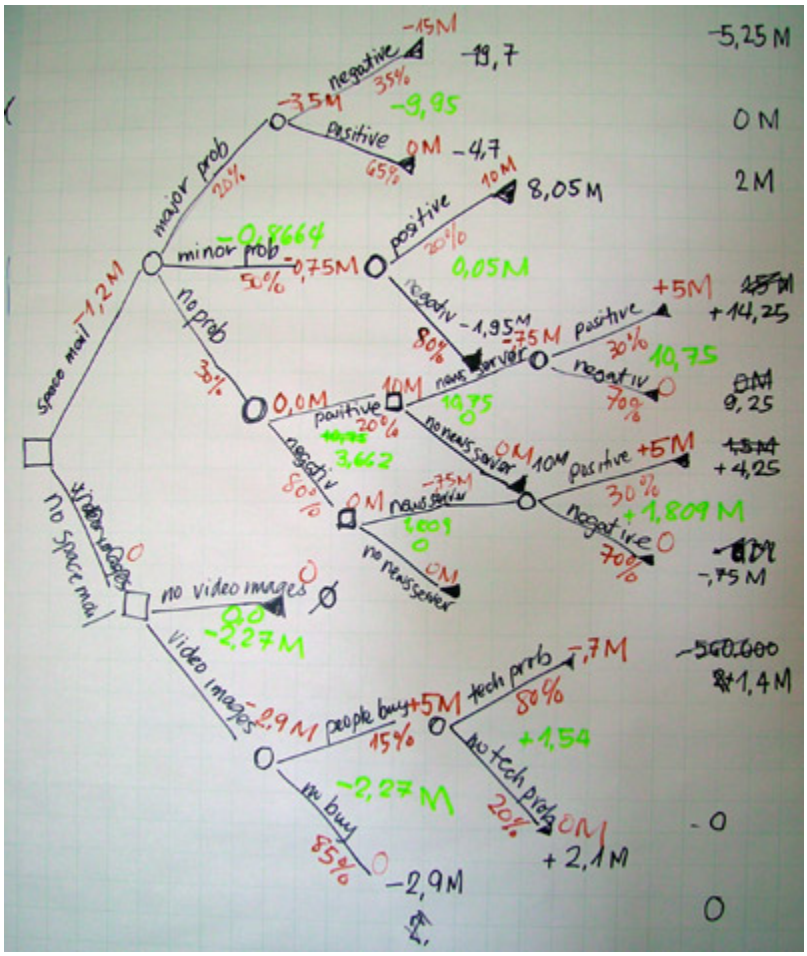
Decision trees

Raport teoretic

Prezenare generală

În general, un arbore de decizie este o structură asemănătoare unei diagrame de flux în care fiecare nod intern reprezintă un „test” asupra unui atribut (de exemplu, dacă o monedă se ridică cu cap sau cozi), fiecare ramură reprezintă rezultatul testului și fiecare nod frunză reprezintă un etichetă de clasă (decizie luată după calcularea tuturor atributelor).



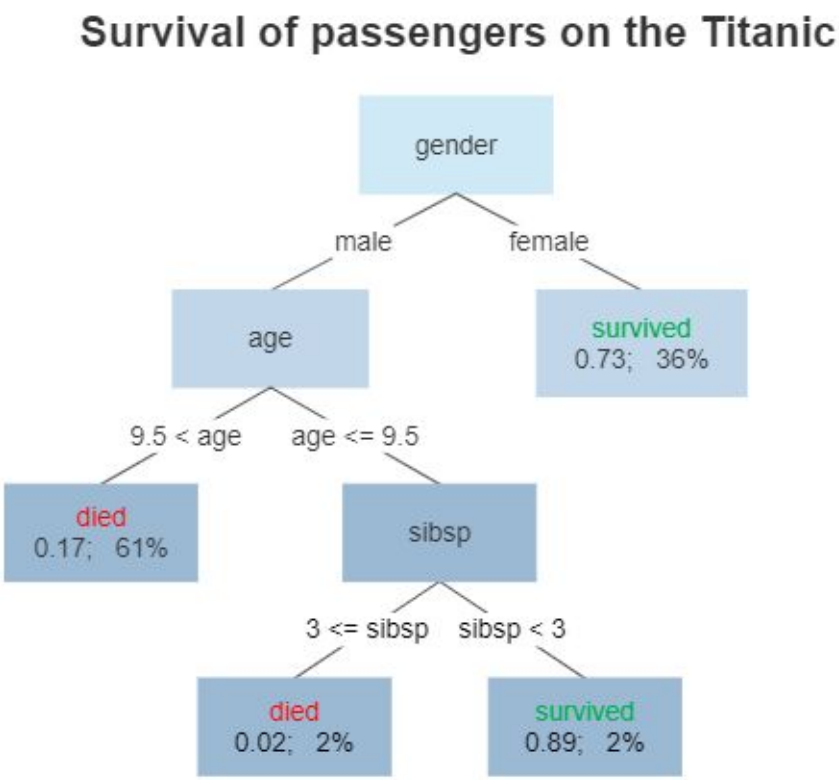
Căile de la rădăcină la frunză reprezintă reguli de clasificare.

În analiza deciziei, un arbore de decizie și diagrama de influență strâns legată sunt utilizate ca instrument vizual și analitic de sprijin pentru decizii, unde sunt calculate valorile așteptate (sau utilitatea așteptată) ale alternativelor concurente.

Un arbore de decizie este format din trei tipuri de noduri:[2]

- Nodurile de decizie – reprezentate de obicei prin pătrate
- Nodurile șansă – reprezentate de obicei prin cercuri
- Nodurile finale – reprezentate de obicei prin triunghiuri

Arborele de decizie sunt utilizați în mod obișnuit în cercetarea operațională și managementul operațiunilor. Dacă, în practică, deciziile trebuie luate online fără rechemare sub cunoștințe incomplete, un arbore de decizie ar trebui să fie paralel cu un model de probabilitate ca model de cea mai bună alegere sau algoritm de model de selecție online. O altă utilizare a arborilor de decizie este ca un mijloc descriptiv pentru calcularea probabilităților condiționate.



În psihologie, metodele arborelui de decizie au fost folosite pentru a modela conceptul uman de învățare. Pe parcurs, cercetătorii au descoperit că algoritmul era util pentru programare. În 1972, primul arbore de clasificare a apărut în proiectul THAID al lui Messenger și Mandell.

Arbori de decizie în informatică

Învățarea arborelui decizional este o abordare de învățare supravegheată (sunt necesare label-uri pentru antrenarea modelului) utilizată în statistici, data mining și învățarea automată. În acest formalism, un arbore de decizie de clasificare sau regresie este utilizat ca model predictiv pentru a trage concluzii despre un set de observații, scopul fiind de a crea un model care prezice valoarea unui atribut pe baza mai multor atribute de intrare.

Modelele de arbore în care variabila țintă poate lua un set discret de valori sunt numite arbori de clasificare ; un arbore de decizie sau un arbore de clasificare este un arbore în care fiecare nod intern (fără frunze) este etichetat cu o caracteristică de intrare.

Arcurile care provin de la un nod etichetat cu o caracteristică de intrare sunt etichetate cu fiecare dintre valorile posibile ale caracteristicii țintă sau arcul duce la un nod de decizie subordonat pe o caracteristică de intrare diferită. Fiecare frunză a arborelui este etichetată cu o clasă sau o distribuție de probabilitate peste clase, ceea ce înseamnă că setul de date a fost clasificat de arbore fie într-o anumită clasă, fie într-o anumită distribuție de probabilitate (care, dacă arborele de decizie este bine-construit, este înclinat către anumite subseturi de clase).

Arborele de decizie se numără printre cei mai populari algoritmi de învățare automată, având în vedere inteligibilitatea și simplitatea lor.

În analiza deciziei, un arbore de decizie poate fi folosit pentru a reprezenta vizual și explicit deciziile și luarea deciziilor.

În data mining, un arbore de decizie descrie datele (dar arborele de clasificare rezultat poate fi o intrare pentru luarea deciziilor).

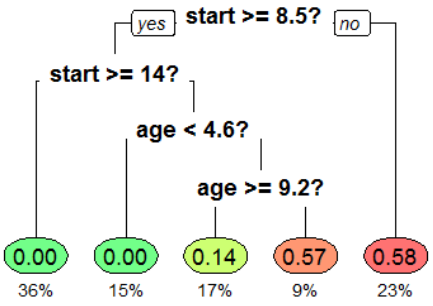
Un arbore este construit prin împărțirea setului original, constituind nodul rădăcină al arborelui, în subseturi – care constituie copiii succesori. Împărțirea se bazează pe un set de reguli de împărțire bazate pe caracteristicile de clasificare. Acest proces se repetă pe fiecare subset derivat într-o manieră recursivă numită partiționare recursivă. Algoritmul este finalizată atunci când subsetul de la un nod are toate aceleași valori ale variabilei țintă sau când divizarea nu mai adaugă valoare predicțiilor sau când informația câștigată în urma diviziunii nu satisface o regulă impusă de scriitori algoritmului. Acest proces de inducere de sus în jos a arborilor de decizie este un exemplu de algoritm lacom (greedy algorithm) și este de departe cea mai comună strategie de învățare a arborilor de decizie din date.

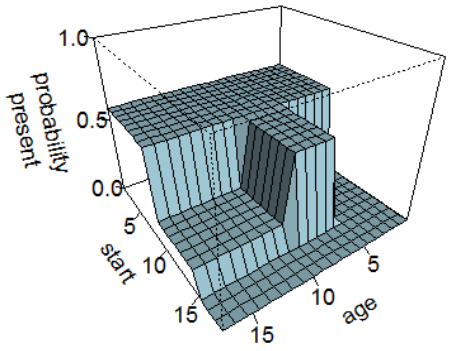
În data mining, arborii de decizie pot fi descriși și ca o combinație de tehnici matematice și de calcul pentru a ajuta la descrierea, clasificarea și generalizarea unui anumit set de date.

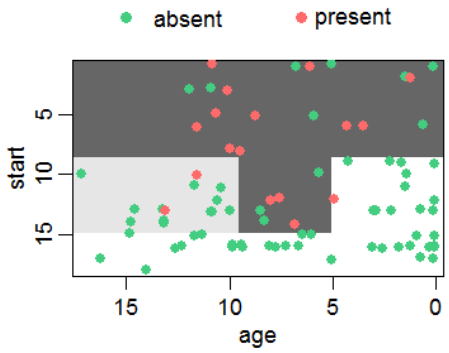
Datele pe care le avem la îndemână sunt procesate în forma următoare pentru a putea fi cât mai ușor de folosit:

$$(\mathbf{x}, Y) = (x_1, x_2, x_3, \ldots, x_k, Y)$$

Variabila dependentă Y este variabila țintă pe care încercăm să o înțelegem, să o clasificăm sau să o generalizăm. Vectorul x este compus din atributele x_1, x_2,... (etc), care sunt utilizate pentru sarcina de clasificare/antrenare.







Tipuri de arbore de decizie

Arborii de decizie utilizați în data mining sunt de două tipuri principale:

- Analiza arborelui de clasificare este atunci când rezultatul prezis este clasa (discretă) căreia îi aparțin datele.
- Analiza arborelui de regresie este atunci când rezultatul prezis poate fi considerat un număr real (de exemplu, prețul unei case sau durata șederii unui pacient într-un spital).

Termenul de analiză a arborelui de clasificare și regresie este un termen umbrelă folosit pentru a se referi la oricare dintre procedurile de mai sus, introdus pentru prima dată de Breiman și colab. în 1984. Arborii utilizați pentru regresie și arborii utilizați pentru

clasificare au unele asemănări – dar și unele diferențe, cum ar fi procedura utilizată pentru a determina unde să se despartă.

Unele tehnici, adesea numite metode de ansamblu, construiesc mai mult de un arbore de decizie:

- Boosted trees. Construirea progresivă a unui ansamblu prin antrenarea fiecărei instanțe noi pentru a sublinia instanțele de antrenament modelate greșit anterior. Un exemplu tipic este AdaBoost. Acestea pot fi folosite pentru probleme de tip regresie și de tip clasificare.
- Arborii de decizie Bootstrap agregați (sau în sac), o metodă de ansamblu timpurie, construiește mai mulți arbori de decizie prin reeșantionarea în mod repetat a datelor de antrenament cu înlocuire și votând arborii pentru o predicție de consens.
- Un clasificator de pădure aleatoriu (random forest) este un tip specific de agregare bootstrap
- Pădure de rotație – în care fiecare arbore de decizie este antrenat prin aplicarea mai întâi a analizei componentelor principale pe un subset aleatoriu al caracteristicilor de intrare.
- Un caz special al unui arbore de decizie este o listă de decizii, care este un arbore de decizie unilateral, astfel încât fiecare nod intern are exact 1 nod frunză și exact 1 nod intern ca copil (cu excepția nodului cel mai de jos, al cărui singurul copil este un nod cu o singură frunză). Deși sunt mai puțin expresive, listele de decizii sunt, fără îndoială, mai ușor de înțeles decât arborii de decizie generali, datorită dispersității lor adăugate, permit impunerea unor metode de învățare non-lacomi și constrângeri monotone.

Exemple notabile de arbori de decizie includ:

- ID3 (dihotomizor iterativ 3)
- C4.5 (succesorul ID3)
- CART (Arborele de clasificare și regresie)
- Detectarea automată a interacțiunii chi-pătrat (CHAID). Efectuează împărțiri pe mai multe niveluri atunci când calculează arbori de clasificare.
- MARS: extinde arborii de decizie pentru a gestiona mai bine datele numerice.
- Arbori de inferență condiționată. Abordare bazată pe statistici care utilizează teste non-parametrice ca criterii de împărțire, corectate pentru teste multiple pentru a

evita supraadaptarea. Această abordare are ca rezultat o selecție imparțială a predictorilor și nu necesită tăiere.

ID3 și CART au fost inventate independent aproximativ în același timp (între 1970 și 1980), dar urmează o abordare similară pentru învățarea unui arbore de decizie din tupluri de antrenament.

S-a propus, de asemenea, să se folosească conceptele teoriei mulțimilor fuzzy pentru definirea unei versiuni speciale a arborelui de decizie, cunoscută sub numele de Arborele de decizie fuzzy (FDT). În acest tip de clasificare fuzzy, în general, un vector de intrare x este asociat cu mai multe clase, fiecare cu o valoare de încredere diferită. De asemenea, ansamblurile amplificate de FDT au fost investigate recent și au arătat performanțe comparabile cu cele ale altor clasificatoare fuzzy foarte eficienți.

Metoda de stabilire a nodului criteriu

Pentru a stabili ce criteriu vom pune ca nod pentru un anumit nod, scopul este să câștigăm cât mai multă informație cu criteriul pe care îl punem.

Entropia este definită după cum urmează:

$$\mathrm{H}(T) = \mathrm{I}_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

Unde p_1, p_2,... sunt fracții care însumează 1 și reprezintă procentul fiecărei clase prezente în nodul copil care rezultă dintr-o împărțire în arbore.

$$\overbrace{IG(T,a)}^{\text{information gain}} = \overbrace{\mathrm{H}(T)}^{\text{entropy (parent)}} - \overbrace{\mathrm{H}(T \mid a)}^{\text{sum of entropies (children)}}$$

$$= -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{i=1}^{J} -\Pr(i \mid a) \log_2 \Pr(i \mid a)$$

Făcând media asupra valorilor posibile ale lui A,

$$\overbrace{E_A(\mathrm{IG}(T,a))}^{\text{expected information gain}} = \overbrace{I(T;A)}^{\text{mutual information between } T \text{ and } A} = \overbrace{\mathrm{H}(T)}^{\text{entropy (parent)}} - \overbrace{\mathrm{H}(T \mid A)}^{\text{weighted sum of entropies (children)}}$$

$$= -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{a} p(a) \sum_{i=1}^{J} -\Pr(i \mid a) \log_2 \Pr(i \mid a)$$

Unde suma ponderată a entropiilor este dată de:

$$\mathrm{H}(T \mid A) = \sum_{a} p(a) \sum_{i=1}^{J} -\Pr(i \mid a) \log_2 \Pr(i \mid a)$$

Adică, câștigul de informații așteptat este informația reciprocă, ceea ce înseamnă că, în medie, reducerea entropiei lui T este informația reciprocă.

Câștigul de informații este folosit pentru a decide pe ce caracteristică să se împartă la fiecare pas în construirea arborelui. Simplitatea este cea mai bună, așa că vrem să ne păstrăm copacul mic. Pentru a face acest lucru, la fiecare pas ar trebui să alegem diviziunea care are ca rezultat cele mai consistente noduri copil. O măsură de consistență folosită în mod obișnuit se numește informație. Pentru fiecare nod al arborelui, valoarea informației „reprezintă cantitatea așteptată de informații care ar fi necesară pentru a specifica dacă o instanță nouă ar trebui clasificată da sau nu, având în vedere că exemplul a ajuns la acel nod”.

Avantaje

Printre alte metode de data mining, arborii de decizie au diverse avantaje:

- Simplu de înțeles și interpretat. Oamenii sunt capabili să înțeleagă modelele arborelui de decizie după o scurtă explicație. Arborii pot fi, de asemenea, afișați grafic într-un mod ușor de interpretat de către cei care nu sunt experți.
- Capabil să gestioneze atât date numerice, cât și categoriale.
- Alte tehnici sunt de obicei specializate în analiza seturi de date care au un singur tip de variabilă. (De exemplu, regulile de relație pot fi utilizate numai cu variabile nominale, în timp ce rețelele neuronale pot fi utilizate numai cu variabile numerice sau categoriile convertite în valori 0-1.)
- Necesită puțină preprocesare a datelor. Alte tehnici necesită adesea normalizarea datelor.
- Utilizează un model “white box” sau “open box”. Dacă o situație dată este observabilă într-un model, explicația condiției este ușor de explicat prin logica booleană. În schimb, într-un model “black box”, explicația rezultatelor este de obicei dificil de înțeles, de exemplu cu o rețea neuronală artificială.
- Posibilitatea de a valida un model folosind teste statistice. Acest lucru face posibilă luarea în considerare a fiabilității modelului.

- Abordare non-parametrică care nu face ipoteze privind datele de antrenament sau reziduurile de predicție; de exemplu, fără ipoteze de distribuție, independență sau variantă constantă
- Funcționează bine cu seturi mari de date. Cantități mari de date pot fi analizate folosind resurse de calcul standard într-un timp rezonabil.
- Precizie cu modelare flexibilă. Aceste metode pot fi aplicate cercetării în domeniul sănătății cu o acuratețe sporită.
- Oglindește luarea deciziilor umane mai îndeaproape decât alte abordări. Acest lucru ar putea fi util atunci când se modelează deciziile/comportamentul uman.
- Robuste împotriva co-liniarității, în special a creșterii.
- În selecția caracteristicilor integrate. Caracteristicile suplimentare irelevante vor fi mai puțin utilizate, astfel încât să poată fi eliminate la rulările ulterioare. Ierarhia atributelor dintr-un arbore de decizie reflectă importanța atributelor. Înseamnă că caracteristicile de sus sunt cele mai informative.
- Arborii de decizie pot aproxima orice funcție booleană, de exemplu XOR.

Limitări

- Copacii pot fi foarte greu adaptibili incremental. O mică modificare a datelor de antrenament poate duce la o schimbare mare a arborelui și, în consecință, a predicțiilor finale.
- Problema învățării unui arbore de decizie optim este cunoscută a fi NP-completă sub mai multe aspecte ale optimității și chiar pentru concepte simple. În consecință, algoritmii practici de învățare a arborelui de decizie se bazează pe euristici, cum ar fi algoritmul lacom, în care deciziile optime la nivel local sunt luate la fiecare nod. Astfel de algoritmi nu pot garanta returnarea arborelui de decizie optim la nivel global.
- Pentru anumite date, arborele de decizie pot deveni arbori prea complexi care nu se generalizează bine din datele de antrenare. (Acest lucru este cunoscut sub numele de overfitting.) Mecanisme precum tăierea (pruning) sunt necesare pentru a evita această problem.
- Adâncimea medie a arborelui, care este definită de numărul de noduri sau de teste până la clasificare, nu este garantată a fi minimă sau mică în baza diferitelor criterii de împărțire.

Referinte

[Decision tree - Wikipedia]

Raport Experimental

Studiu de caz 1

Using Tree-Based Machine Learning for Health Studies: Literature Review and Case Series

Liangyuan Hu1,* and Lihua Li2

Paul B. Tchounwou, Academic Editor

Using Tree-Based Machine Learning for Health Studies: Literature Review and Case Series - PMC (nih.gov)

Obiectivul lucrării

Obiectivul lucrării este de a încuraja utilizarea metodelor bazate pe arbori în cercetarea în sănătate. Autorii revizuiesc fundamentele metodologice a trei metode cheie de învățare automată bazate pe arbori: păduri aleatorii, "extreme gradient boosting" și arbori de regresie aditivă bayesiană. Ei aplică metode de ansamblu de arbori pentru a selecta predictori importanți pentru prezența complicațiilor respiratorii postoperatorii în rândul pacienților cu cancer pulmonar în stadiu incipient cu tumori rezecabile. Apoi demonstrează cum se folosesc de anumite metode pentru a estima efectele cauzale ale abordărilor chirurgicale populare asupra complicațiilor respiratorii postoperatorii în rândul pacienților cu cancer pulmonar. Folosind aceleași date, au implementat metodele pentru a estima cu acuratețe ponderile de probabilitate inversă pentru o analiză a scorului de propensitate a eficacității comparative a abordărilor chirurgicale. În cele din urmă, au demonstrat cum pot fi folosite pădurile aleatorii pentru a imputa datele lipsă folosind setul de date Study of Women's Health Across the Nation.

Autorii s-au inspirat în alegerea de a folosi arbori de decizie datorită existenței aplicațiilor care arată că metodele bazate pe arbore pot genera rezultate mai bune decât metodele tradiționale. Pentru a numi câteva, descoperirea biomarkerilor în studiile proteomice [1], estimarea efectelor cauzale [2,3], predicția costului asistenței medicale [4], identificarea factorilor cheie de risc [5,6] și evaluarea performanței spitalului [7] . În studiile de sănătate, metodele bazate pe arbori nu au câștigat aceeași tracțiune ca în știința datelor.

Datele folosite

Datele prelucrate în experimentele efectuate au fost obținute din diverse surse, cum ar fi Sondajul Național de Sănătate și Nutriție Examination Survey (NHANES), National Health Interview Survey (NHIS) și National Survey on Drug Use and Health

(NSDUH). Autorii au folosit aceste surse de date pentru a demonstra modul în care metodele de învățare automată bazate pe arbore pot fi folosite pentru a rezolva probleme importante de sănătate.

Setul de date de analiză a inclus 3302 femei cu vârsta cuprinsă între 42 și 52 de ani, care au fost înscrise în 1996-1997 din șapte site-uri din SUA și au fost urmărite până în 2018 anual. Există un interes puternic în utilizarea datelor SWAN pentru a identifica factorii cheie de risc pentru rezultatele sănătății, cum ar fi sindromul metabolic. Cu toate acestea, o problemă dificilă este prezența datelor lipsă. Dintre 60 de variabile potențiale predictoare, doar 11 variabile au fost observate în totalitate; cantitatea de date lipsă în variabile a variat între 0,1% și 27,1%.

Rezultate

Rezultatele obținute din studiile de caz au arătat că metodele de învățare automată bazate pe arbore pot fi folosite pentru a rezolva probleme importante de sănătate. De exemplu, ele pot fi utilizate pentru a prezice riscul de a dezvolta diabet de tip 2.

Autorii au folosit mai multe metode:

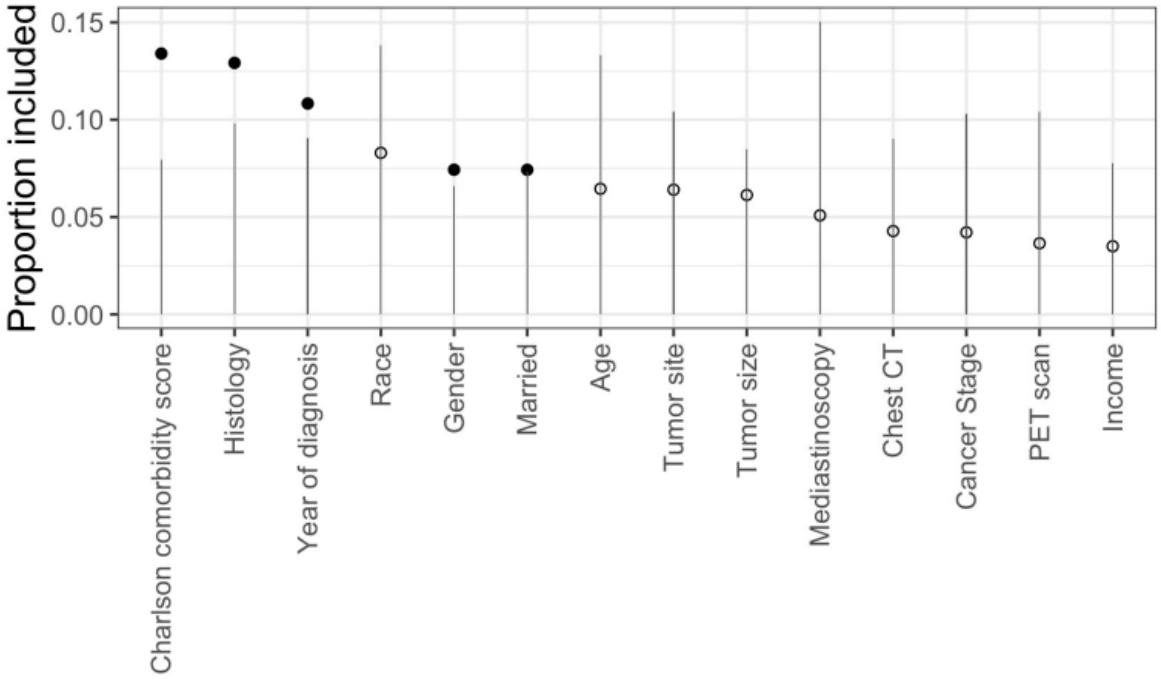
- CART
- Random forest
- XGBoost
- BART

Datele alese pentru unele dintre metodele de mai sus se afla în următorul tabel:

Methods	Selected Variables	AUC
BART	Chalson comorbidity score, gender, married, histology, year of diagnosis	0.85
XGBoost	Age, year of diagnosis	0.72
RF	Chalson comorbidity score, histology	0.74

AUC este abrevierea pentru ”area under the receiver operating characteristics curve”, ceea ce indică gradul de separare al claselor.

Selecția variabilelor pentru modelul BART este vizualizată în următorul tabel:



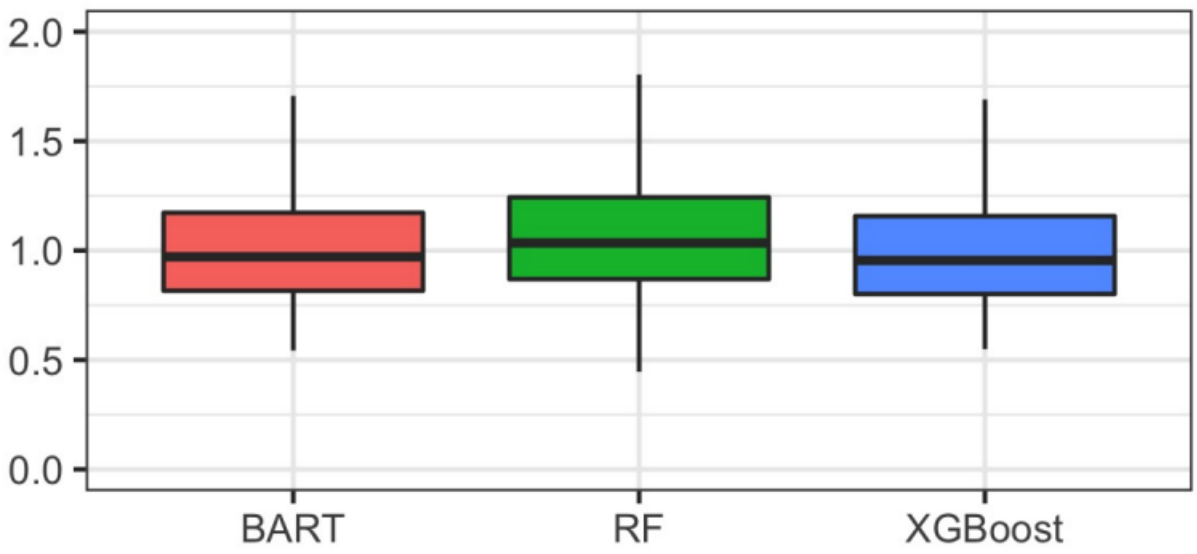
Arborii de decizie au fost folosiți și la inferențe cauzale despre efectele medii ale tratamentului a trei abordări chirurgicale asupra complicațiilor respiratorii postoperatorii bazate pe riscul relativ, folosind datele de cancer pulmonar SEER-Medicare.

Intervalele de incertitudine de 95% sunt afișate în paranteze. Au fost utilizați toți cei 14 potențiali factori de confuzie.

RAS: chirurgie robotizată; VATS: chirurgie toracică video-asistată; OT: toracotomie deschisă.

Methods	RAS vs. OT	RAS vs. VATS	OT vs. VATS
BART	0.94 (0.72, 1.16)	1.09 (0.84, 1.34)	1.12 (0.87, 1.37)
XGBoost	0.91 (0.64, 1.13)	1.04 (0.79, 1.28)	1.08 (0.84, 1.33)
RF	0.90 (0.63, 1.14)	1.03 (0.78, 1.29)	1.06 (0.82, 1.35)

Următoarea figură arată distribuția probabilității inverse a ponderilor de tratament estimate prin fiecare dintre cele trei metode bazate pe arbore. Rețineți că media posterioară a greutăților a fost utilizată pentru BART. Greutățile estimate din cele trei metode au avut distribuții similare și nu au existat ponderi extreme care să fie îngrijorătoare



1. Hernández et al., Bayesian methods for proteomic biomarker development. EuPA Open Proteom., 2015;9:54-64. doi: [10.1016/j.euprot.2015.06.001](https://doi.org/10.1016/j.euprot.2015.06.001)

2. Hu et al., Estimation of causal effects of multiple treatments in observational studies with a binary outcome. Stat Methods Med Res., 2020;29(11):3218-3234 doi: [10.1177/0962280220921909](https://doi.org/10.1177/0962280220921909)

3. Hu and Gu., Estimation of causal effects of multiple treatments in healthcare database studies with rare outcomes. Health Serv Outcomes Res Methodol., 2021;21(4):287-308 doi: [10.1007/s10742-020-00234-4](https://doi.org/10.1007/s10742-020-00234-4)

4. Mazumdar et al., Comparison of statistical and machine learning models for healthcare cost data: A simulation study motivated by Oncology Care Model (OCM) data BMC Health Serv Res., 2020;20(1):350 doi: [10.1186/s12913-020-05275-w](https://doi.org/10.1186/s12913-020-05275-w)

5. Hu et al., Tree-Based Machine Learning to Identify and Understand Major Determinants for Stroke at the Neighborhood Level J Am Heart Assoc., 2020;9(24):e016745 doi: [10.1161/JAHA.120.016745](https://doi.org/10.1161/JAHA.120.016745)

6.Hu et al., Ranking sociodemographic, health behavior, prevention, and environmental factors in predicting neighborhood cardiovascular health: A Bayesian machine learning approach Prev Med., 2020;141:106240 doi: [10.1016/j.jclinepi.2021.03.017](https://doi.org/10.1016/j.jclinepi.2021.03.017)

7.Liu et al., Ensemble of trees approaches to risk adjustment for evaluating a hospital's performance Health Care Manag Sci., 2015;18(1):58-66 doi: [10.1007/s10729-014-9283-x](https://doi.org/10.1007/s10729-014-9283-x)

Studiu de caz 2

An analysis of boosted ensembles of binary fuzzy decision trees

Marco Barsacchi, Alessio Bechini* , Francesco Marcelloni

Dept. of Information Engineering, University of Pisa, Largo L. Lazzarino, Pisa 56122, Italy

Obiectivele lucrării

Principala contribuție a lucrării constă în prima descriere detaliată şi cuprinzătoare a abordării FDT-Boost, împreună cu o investigaţie experimentală asupra caracteristicilor a procesului său de învățare şi a capacității sale de a oferi rezultate precise şi robuste, într-o comparație încrucișată cu alți clasificatori fuzzy de ultimă generație. Robustitatea în acest context este concepută ca o sensibilitate scăzută la zgomotul etichetei. În evaluarea procesului de învățare se propune o abordare inedită bazată pe inspectarea tendinței entropiei de greutate, alături de altele mai tradiționale. Un alt rezultat este: complexitatea modelelor generate de FDT-Boost este mai mică decât omologii lor non-fuzzy, create prin utilizarea SAMME-AdaBoost cu arbori de decizie binari clasici multiclasă. Acest lucru face ca modelele amplificate să fie mai potrivite pentru a fi implementate în sisteme de calcul cu constrângeri de memorie.

Datele folosite

Pentru evaluarea performanței algoritmilor de clasificare autorii au folosit doar seturi de date extrase din depozitul Keel (hxxp://sci2s.ugr[.]es/keel/datasets.php).

Câteva atribute utilizate:

Table 1
Datasets in the "tuning benchmark" for FDT-Boost, and their characteristics.

Dataset	Cardinality	# Attr. (R,I,C)	# Classes
automobile	150	25 (15,0,10)	6
balance	625	4 (4,0,0)	3
bupa	345	6 (1,5,0)	2
ecoli	336	7 (7,0,0)	8
ionosphere	351	33 (32,1,0)	2
saheart	462	9 (5,3,1)	2
sonar	208	60 (60,0,0)	2

Table 2
Datasets in the "evaluation benchmark" for FDT-Boost, and their characteristics.

Dataset	Cardinality	# Attr. (R,I,C)	# Classes	*IR*
appendicitis (APP)	106	7 (7,0,0)	2	4.05
australian (AUS)	690	14 (3,5,6)	2	1.25
bands (BAN)	365	19 (13,6,0)	2	1.70
dermatology (DER)	358	34 (0,34,0)	6	5.55
glass (GLA)	214	9 (9,0,0)	7	7.78
hayes (HAY)	160	4 (0,4,0)	3	2.10
iris (IRI)	150	4 (4,0,0)	3	1.00
magic (MAG)	19,020	10 (10,0,0)	2	1.84
mammographic (MAM)	830	5 (0,5,0)	2	1.06
newthyroid (NEW)	215	5 (4,1,0)	3	5.00
ring (RIN)	7400	20 (20,0,0)	2	1.02
segment (SEG)	2310	19 (19,0,0)	7	1.00
tae (TAE)	151	5 (0,5,0)	3	1.06
vehicle (VEH)	846	18 (0,18,0)	4	1.07
vowel (VOW)	990	13 (10,3,0)	11	1.00
wdbc (WDC)	569	30 (30,0,0)	2	1.68
wine (WIN)	178	13 (13,0,0)	3	1.48
wisconsin (WIS)	683	9 (0,9,0)	2	1.86

Următorul pseudocod este cel de la algoritmul FDT:

Function `LearnFDT`$(\bar{TR}, \mathbf{P}, \beta, n_{min}, \gamma, \eta)$**:**
  Create a *root* node with all the set of data $\widehat{TR}$, i.e. a fuzzy set of all the data, with all the membership values = 1.
  $tree \leftarrow$ `BuildTree`$(node, \widehat{TR}, \mathbf{P}, \beta, n_{min}, \gamma, \eta)$
  return *tree*

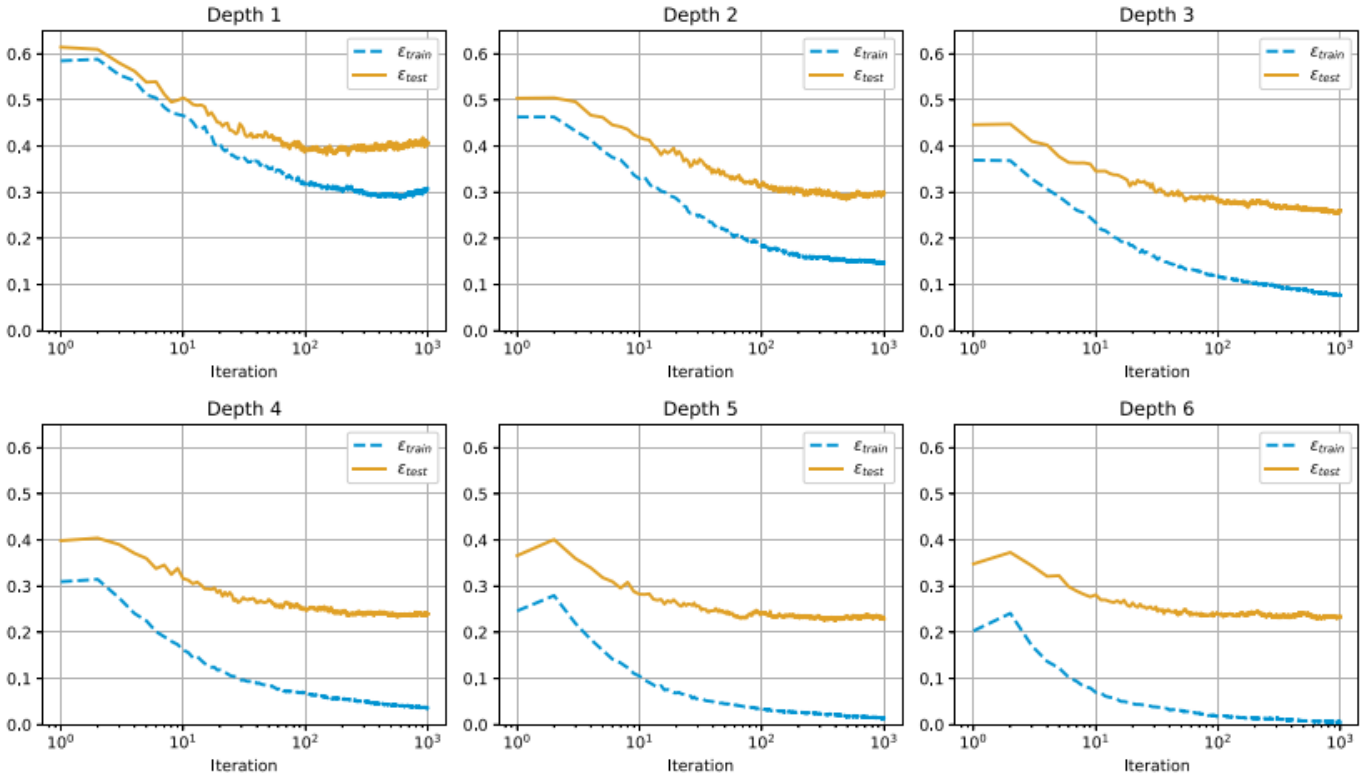
Function `BuildTree`$(node, X, \mathbf{P}, \beta, n_{min}, \gamma, \eta)$**:**
  if `StoppingCondition`$(node, \beta, n_{min}, \gamma, \eta)$ **then**
    $node \leftarrow$ mark *node* as *leaf*
  else
    $splits \leftarrow$ `SelectSplit`$(X, \mathbf{P})$
    foreach $split_k \in splits$ **do**
      $child_k, X_k \leftarrow$ `Membership`$(X, \mathbf{P}, split_k)$
      $child_k \leftarrow$ `BuildTree`$(child_k, X_k, \mathbf{P}, \beta, n_{min}, \gamma, \eta)$
      connect $child_k$ with *node*
    end
  end
  return *node*

Iar următorul este de la FDT Boosted:

Function `FDTBoost`$(TR, T, \beta, n_{min}, \gamma, \eta, s_{max})$**:**
  $\mathbf{P} \leftarrow$ `FuzzyDiscretizer`$(\mathbf{X}, TR, s_{max})$
  $BoostEns \leftarrow \varnothing$
  $\mathbf{w} = \{w_i \leftarrow 1/N, i = 1, \ldots N\}$
  `/* Initially uniform sample weight vector */`
  for $t \leftarrow 1$ **to** T **do**
    $TR^{(t)} \leftarrow$ `WeightedSampling`$(TR, \mathbf{w})$
    `/* TR is sampled according to weight vector` $\mathbf{w}$ `*/`
    $FDT^{(t)} \leftarrow$ `BuildTree`$(TR^{(t)}, \mathbf{P}, \beta, n_{min}, \gamma, \eta)$
    `/* Construction of the` t`-th Fuzzy Decision Tree */`
    $eer^{(t)} \leftarrow$ computation of Eq. 11
    $\alpha^{(t)} \leftarrow$ computation of Eq. 12
    `/* Tree weight` $\alpha^{(t)}$ `depends on error rate` $eer^{(t)}$ `*/`
    $BoostEns \leftarrow BoostEns \cup (FDT^{(t)}, \alpha^{(t)})$
    $\mathbf{w} = \{w_i \leftarrow$ value as per Eq. 13, $i = 1, \ldots N\}$
    $\mathbf{w} \leftarrow \mathbf{w} / \sum_{i=1}^{N} w_i$
    `/* Sample weight vector updated & normalized */`
  end
  return *BoostEns*

Rezultate

Următoarele grafice descriu raportul mediu de eroare de clasificare față de numărul de iterații peste benchmark-ul de reglare pentru diferite valori ale adâncimii maxime β pentru cursantul de bază.



Următorul tabel conține precizia medie și abaterea standard obținute de FDT-Boost, FURIA, FBDT și FMDT pe setul de testare (cross-validated de 5 ori) cu valori standard pentru hiperparametri.

Datasets	Algorithms			
	FDT-Boost	FURIA	FBDT	FMDT
APP	85.80 ± 7.43	**91.13** ± 0.75	83.07 ± 2.14	82.16 ± 8.43
AUS	**84.49** ± 0.35	83.30 ± 6.40	83.91 ± 1.34	84.20 ± 2.57
BAN	**74.51** ± 1.28	74.25 ± 1.97	64.89 ± 2.83	68.79 ± 2.63
DER	**97.78** ± 2.23	84.19 ± 4.87	94.69 ± 2.20	93.01 ± 1.88
GLA	74.31 ± 6.73	**84.95** ± 2.70	66.79 ± 5.22	71.95 ± 8.66
HAY	**85.00** ± 5.00	74.88 ± 2.54	73.75 ± 8.29	63.13 ± 6.67
IRI	**94.67** ± 2.49	93.33 ± 4.71	94.00 ± 3.89	94.00 ± 4.42
MAG	**85.56** ± 0.67	84.78 ± 0.52	79.59 ± 0.70	80.08 ± 0.72
MAM	**83.59** ± 1.63	79.01 ± 6.30	80.55 ± 2.00	80.44 ± 1.43
NEW	**94.88** ± 3.42	94.42 ± 2.37	93.02 ± 2.94	93.02 ± 1.47
RIN	**94.65** ± 0.78	83.69 ± 6.89	83.71 ± 6.81	87.04 ± 1.11
SEG	**96.80** ± 1.05	84.48 ± 4.67	96.79 ± 0.63	95.67 ± 0.51
TAE	**53.01** ± 7.48	45.63 ± 5.06	48.30 ± 7.71	51.61 ± 4.87
VEH	**72.21** ± 3.11	68.09 ± 1.78	70.92 ± 1.49	69.50 ± 1.71
VOW	86.46 ± 2.93	79.80 ± 2.09	79.60 ± 0.88	**94.44** ± 1.59
WDC	**97.18** ± 1.02	95.96 ± 2.12	94.38 ± 1.62	94.55 ± 2.03
WIN	**98.87** ± 1.38	93.78 ± 3.37	91.51 ± 5.73	94.38 ± 3.54
WIS	**97.01** ± 0.79	94.72 ± 7.18	95.32 ± 1.06	95.03 ± 1.23

Aplicația software

Scop

Predicția, utilizând algoritmul de creare a arborilor de decizie, implementat în python 3.11 și analiza datelor și a diferitelor contexte în care arborii de decizie sunt limitați de modul de procesare.

Alegerea datelor

Am ales datasetul Multiple Sclerosis, Disease Conversion Predictors of Clinically Isolated Syndrome(CIS) to Multiple Sclerosis: https://www.kaggle.com/datasets/desalegngeb/conversion-predictors-of-cis-to-multiple-sclerosis

Atributele datelor sunt:

- **ID**: identificatorul pacientului (int)
- **Vârsta**: vârsta pacientului (în ani)
- **Școlarizare**: timpul petrecut de pacient la școală (în ani)
- **Sex**: 1=masculin, 2=feminin
- **Alăptarea**: 1=da, 2=nu, 3=necunoscut
- **Varicela**: 1=pozitiv, 2=negativ, 3=necunoscut
- **Simptome_inițiale**: 1 = vizual, 2 = senzorial, 3 = motor, 4 = altele, 5 = vizual și senzorial, 6 = vizual și motor, 7 = vizual și altele, 8 = senzorial și motor, 9 = senzorial și altele, 10 = motor și altele, 11=Vizual, senzorial și motor, 12=vizual, senzorial și altele, 13=Vizual, motor și altele, 14=Senzorial, motor și altele, 15=vizual, senzorial,motor și altele
- **Mono_sau_polisimptomatic**: 1=monosimptomatic, 2=polisimptomatic, 3=necunoscut
- **Benzile_oligoclonale**: 0=negativ, 1=pozitiv, 2=necunoscut
- **LLSSEP**: 0=negativ, 1=pozitiv
- **ULSSEP**:0=negativ, 1=pozitiv
- **VEP**:0=negativ, 1=pozitiv
- **BAEP**: 0=negativ, 1=pozitiv
- **RMN_periventricular**:0=negativ, 1=pozitiv
- **RMN_cortical**: 0=negativ, 1=pozitiv
- **Infratentorial_MRI**:0=negativ, 1=pozitiv

- **RMN_Spinal_Maduvei**: 0=negativ, 1=pozitiv
- **initial_EDSS**:?
- **final_EDSS**:?
- **Grupa**: 1=CDMS, 2=non-CDMS

BAEP: În neuroanatomia umană, potențialele evocate auditive ale trunchiului cerebral (BAEP), numite și răspunsuri evocate auditive ale trunchiului cerebral (BAER), sunt potențiale evocate auditive foarte mici ca răspuns la un stimul auditiv, care sunt înregistrate de electrozii plasați pe scalp.

VEP: Potențialul evocat vizual (VEP) este un potențial evocat declanșat prin prezentarea unui fulger luminos sau a unui model de stimul care poate fi utilizat pentru a confirma deteriorarea căii vizuale, inclusiv retina, nervul optic, chiasma optică, radiațiile optice și cortexul occipital.

Benzi oligoclonale: benzile oligoclonale (OCB) sunt benzi de imunoglobuline care sunt observate atunci când se analizează serul sanguin sau lichidul cefalorahidian (LCR) al unui pacient.

Sunt utilizate în diagnosticul diferitelor boli neurologice și de sânge.

Benzile oligoclonale sunt prezente în LCR la mai mult de 95% dintre pacienții cu scleroză multiplă definită clinic.

SSEP: Potențialele evocate somatosenzoriale (SSEP) sunt înregistrate din sistemul nervos central după stimularea nervilor periferici. ULSSEP (SSEP al membrului superior), LLSSEP (SSEP al membrului inferior)

EDSS: Expanded Disability Status Scale (EDSS) este o metodă de cuantificare a dizabilității în scleroza multiplă și de monitorizare a modificărilor nivelului de dizabilitate în timp.

Este utilizat pe scară largă în studiile clinice și în evaluarea persoanelor cu SM.

Numărul de înregistrări în setul de date este: 273.

După preprocesarea datelor: eliminarea valorilor ne-definite (nan), au rămas următoarele atribute: Gender, Age, Schooling, Breastfeeding, Varicella, Initial_Symptom, Mono_or_Polysymptomatic, Oligoclonal_Bands, LLSSEP, ULSSEP, VEP, BAEP, Periventricular_MRI, Cortical_MRI, Infratentorial_MRI, Spinal_Cord_MRI, Initial_EDSS, Final_EDSS, group. Iar numărul de înregistrări a ajuns la 125.

Codul python

Importurile modulelor pe care le vom folosi

```
import numpy as np
import pandas as pd
import random
from enum import Enum, auto
```

Citirea datelor si preprocesarea lor:

```
data =
pd.read_csv("conversion_predictors_of_clinically_isolated_syndrome_to_mult
iple_sclerosis.csv")
data.drop(columns=['Unnamed: 0'], inplace=True)
data = data.dropna()
data['group'] = [random.randint(1, 2) for _ in
range(len(data['group']))]
data
```

Definirea algoritmului cu ajutorul claselor.

```
class Node():
    def __init__(self, feature_index=None, threshold=None,
left=None, right=None, info_gain=None, value=None):
        # for decision node
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.info_gain = info_gain
        # for leaf node
        self.value = value

class ModeType(Enum):
    entropy = auto()
    gini = auto()
class DecisionTreeClassifier():
    def __init__(self, keys, min_samples_split=2, max_depth=2):
        # initialize the root of the tree
        self.root = None

        # stopping conditions
```

```
            self.min_samples_split = min_samples_split
            self.max_depth = max_depth
            self.keys = keys

        def build_tree(self, dataset, curr_depth=0):
            X, Y = dataset[:,:-1], dataset[:,-1]
            num_samples, num_features = np.shape(X)

            # split until stopping conditions are met
            if num_samples>=self.min_samples_split and
curr_depth<=self.max_depth:
                # find the best split
                best_split = self.get_best_split(dataset, num_samples,
num_features)
                # check if information gain is positive
                if best_split["info_gain"]>0:
                    # recur left
                    left_subtree =
self.build_tree(best_split["dataset_left"], curr_depth+1)
                    # recur right
                    right_subtree =
self.build_tree(best_split["dataset_right"], curr_depth+1)
                    # return decision node
                    return Node(best_split["feature_index"],
best_split["threshold"],
                                left_subtree, right_subtree,
best_split["info_gain"])

            # compute leaf node
            leaf_value = self.calculate_leaf_value(Y)
            # return leaf node
            return Node(value=leaf_value)

        def get_best_split(self, dataset, num_samples, num_features):
            ''' function to find the best split '''

            # dictionary to store the best split
            best_split = {}
```

```
            max_info_gain = -float("inf")

            # loop over all the features
            for feature_index in range(num_features):
                feature_values = dataset[:, feature_index]
                possible_thresholds = np.unique(feature_values)
                # loop over all the feature values present in the data
                for threshold in possible_thresholds:
                    # get current split
                    dataset_left, dataset_right = self.split(dataset,
feature_index, threshold)
                    # check if childs are not null
                    if len(dataset_left)>0 and len(dataset_right)>0:
                        y, left_y, right_y = dataset[:, -1],
dataset_left[:, -1], dataset_right[:, -1]
                        # compute information gain
                        curr_info_gain = self.information_gain(y,
left_y, right_y, "gini")
                        # update the best split if needed
                        if curr_info_gain>max_info_gain:
                            best_split["feature_index"] = feature_index
                            best_split["threshold"] = threshold
                            best_split["dataset_left"] = dataset_left
                            best_split["dataset_right"] = dataset_right
                            best_split["info_gain"] = curr_info_gain
                            max_info_gain = curr_info_gain

            # return best split
            return best_split

        def split(self, dataset, feature_index, threshold):
            ''' function to split the data '''

            dataset_left = np.array([row for row in dataset if
row[feature_index]<=threshold])
            dataset_right = np.array([row for row in dataset if
row[feature_index]>threshold])
            return dataset_left, dataset_right
```

```
        def information_gain(self, parent, l_child, r_child,
mode=ModeType.entropy):
            ''' function to compute information gain '''

            weight_l = len(l_child) / len(parent)
            weight_r = len(r_child) / len(parent)
            if mode==ModeType.gini:
                gain = self.gini_index(parent) -
(weight_l*self.gini_index(l_child) + weight_r*self.gini_index(r_child))
            else:
                gain = self.entropy(parent) -
(weight_l*self.entropy(l_child) + weight_r*self.entropy(r_child))
            return gain

        def entropy(self, y):
            ''' function to compute entropy '''

            class_labels = np.unique(y)
            entropy = 0
            for cls in class_labels:
                p_cls = len(y[y == cls]) / len(y)
                entropy += -p_cls * np.log2(p_cls)
            return entropy

        def gini_index(self, y):
            ''' function to compute gini index '''

            class_labels = np.unique(y)
            gini = 0
            for cls in class_labels:
                p_cls = len(y[y == cls]) / len(y)
                gini += p_cls**2
            return 1 - gini

        def calculate_leaf_value(self, Y):
            ''' function to compute leaf node '''
```

```
            Y = list(Y)
            return max(Y, key=Y.count)

        def print_tree(self, tree=None, indent=" "):
            ''' function to print the tree '''

            if not tree:
                tree = self.root

            if tree.value is not None:
                print(tree.value)

            else:
                print(f"[{str(self.keys[tree.feature_index])} <=
{tree.threshold}] IG: {tree.info_gain}")
                print(f"{indent}left:", end="")
                self.print_tree(tree.left, indent + "  ")
                print(f"{indent}right:", end="")
                self.print_tree(tree.right, indent + "  ")

        def fit(self, X, Y):
            ''' function to train the tree '''

            dataset = np.concatenate((X, Y), axis=1)
            self.root = self.build_tree(dataset)

        def predict(self, X):
            ''' function to predict new dataset '''

            preditions = [self.make_prediction(x, self.root) for x in X]
            return preditions

        def make_prediction(self, x, tree):
            ''' function to predict a single data point '''

            if tree.value!=None: return tree.value
            feature_val = x[tree.feature_index]
```

```
            if feature_val<=tree.threshold:
                return self.make_prediction(x, tree.left)
            else:
                return self.make_prediction(x, tree.right)
```

Vom împărți setul de date în două seturi, unul de antrenare și unul de testare, în funcție de un factor de împărțire de 0.8, astfel încât în setul de antrenament vom avea 80% din date, iar în setul de testare 20% din date.

```
    X = data.iloc[:, :-1].values
    Y = data.iloc[:, -1].values.reshape(-1,1)
    from sklearn.model_selection import train_test_split
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=.2, random_state=41)
```

Vom atrena modelul:

```
classifier = DecisionTreeClassifier(keys=data.keys(), min_samples_split=2,
max_depth=20)
classifier.fit(X_train,Y_train)
classifier.print_tree()
```

Testarea setului de testare.

```
Y_pred = classifier.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(Y_test, Y_pred)
```