Učenje stabala odluke

Mauro Raguzin

27. veljače 2023.

Sadržaj

1 Sažetak **1**

2 O učenju stabala odluke **2**
2.1 Uvod i povijest 2
2.2 Izražajnost i ograničenja stabala odluke 2

3 Praktična implementacija učenja **3**
3.1 Odabir najboljeg atributa za testiranje 4
3.2 Ograničenja učenja stabala odluke i moguća poboljšanja . . . 5

4 Opis softverskog rješenja **6**
4.1 Korištenje i struktura programa 6
4.2 Implementirane funckionalnosti 7

5 Primjeri **8**
5.1 Odabir restorana 8
5.2 Zapošljavanje svježih diplomanata 8

6 Zaključak **9**

1 Sažetak

U ovom radu ukratko objašnjavamo jednu od tehnika strojnog učenja s nadzorom — stabla odluke — te prezentiramo izrađeno programsko rješenje koje implementira osnovni algoritam učenja takvih stabala za rješavanje problema booleovske klasifikacije , uz obrazlaganje nekih implementiranih poboljšanja povrh osnovnog algoritma. Konačno, promatramo nekoliko primjera izgrađenih stabala odluke korištenjem ovog programa.

2 O učenju stabala odluke

2.1 Uvod i povijest

Stabla odluke predstavljaju jedan od najstarijih, ali i povijesno najuspješnijih pristupa strojnog učenja. Kao kod bilo koje tehnike učenja s nadzorom, cilj ovog učenja je aproksimirati nepoznatu funkciju hipotezom koju sada želimo prikazati u obliku stabla. Preciznije, na ulazu primamo vektor vrijednosti nekog konačnog broja *atributa*, a na izlazu dajemo jednu vrijednost — *odluku*. Ta odluka može biti element proizvoljnog skupa, ali mi ćemo se ograničiti na aproksimiranje samo jednostavnije klase funkcija čija je domena predstavljena s n atributa (proizvoljne domene, no najčešće su to brojevi i razne kategorije), a kodomena sa $\{\top, \bot\}$. To su tzv. problemi booleovske klasifikacije, gdje svaki primjer za učenje mora biti klasficiran kao istinit (*pozitivan* primjer) ili lažan (*negativan* primjer) [1]. Iz skupa atributa ovog problema učenja izdvajamo *ciljni* atribut koji odgovara klasifikaciji ($\top$ ili $\bot$) danog primjera za učenje.

Naš cilj je pronalazak dovoljno efikasnih algoritama za konstrukciju ovakvih stabala te izgradnja programa koji može učiti na velikim skupovima vektora. Pritom zahtijevamo da je broj primjera za učenje dovoljno velik da pokriva sve vrijednosti kategoričkih atributa — onih koji poprimaju vrijednosti iz relativno malog konačnog skupa — te da dobiveno stablo bude konzistentno tj. da se isti primjeri za učenje, nakon izgradnje stabla, opet podjednako klasificiraju. Neka ograničenja ovog pristupa će biti obrazložena kasnije.

2.2 Izražajnost i ograničenja stabala odluke

Gore opisano stablo je evidentno ekvivalentno tvrdnji da je ciljni atribut istinit ako i samo ako postoji put u stablu odluke od korijena do lista označenog s $\top$ takav da je na svakom unutarnjem čvoru odabrana ona iduća grana koja odgovara vrijednosti odgovarajućeg atributa u ulaznom vektoru. Konciznije, imamo

$$Cilj \Leftrightarrow (Put_1 \vee Put_2 \vee \ldots),$$

gdje je svaki Put konjunkcija odgovarajućih testova atributa na vektoru. Iz poznatih rezultata iz logike sudova možemo zaključiti da je ova formula ekvivalentna nekoj disjunktivnoj normalnoj formi, što znači da se svaka funkcija logike sudova može zapisati kao stablo odluke.

Dakle, izražajnost ovog pristupa učenja teoretski nije zanemariva, iako u praksi ubrzo dolazimo do otprije poznatih problema s logikom sudova: javlja se potreba za uvođenjem jako velikog broja varijabli radi izražavanja naoko

jednostavnih tvrdnji. Tako primjerice već za zapisivanje većinske funkcije trebamo stablo čija veličina eksponencijalno raste s brojem ulaza! Ovaj problem je općenit i ne postoji niti jedna reprezentacija koja bi efikasno reprezentirala sve moguće logičke funkcije, jer tih funkcija na n atributa ima 2^{2^n}.

3 Praktična implementacija učenja

Jasno je da općenito ne postoji efikasan način učenja stabala odluke jer je moguće dobiti stabla veličine bar 2^{2^n}, za kojih nema nade da ćemo ih ikada moći obići. Ipak, primjenom dobrih pohlepnih heurstika, moguće je dobiti sasvim razumna stabla odluke.

Glavni algoritam učenja je rekurzivne prirode te se može promatrati kao plodna kombinacija dvaju klasičnih tehnika dizajna algoritama: pohlepnost i podijeli-pa-vladaj. Naime, problem učenja jednostavno možemo promatrati kao dubinsko pretraživanje, samo što sada *dodajemo* podstabla u svakoj razini rekurzije. Ta podstabla odgovaraju svim mogućim odabirima grane s trenutnog internog čvora, a on predstavlja neki od atributa učenja, dok grane do čvorova-djece predstavljaju jednu od odabranih vrijednosti tog (roditeljskog) atributa. Pri svakom silasku u novo podstablo, moramo kao skup relevantnih primjera u novom potproblemu uzeti *samo* one primjere koji imaju odgovarajuću komponentu vektora jednaku vrijednosti odabrane grane; također, zanemarujemo odabran atribut jer on više ne može biti relevantan u daljnjim odlukama.

Rekurzija jasno u nekom trenutku mora završiti, što se ovdje može dogoditi na četiri posebna načina:

1. Ako su *svi* preostali (pot)primjeri pozitivni ili negativni (k), tada smo gotovi s ovom granom: trenutni čvor je list s klasifikacijom k.

2. Ako nam je preostalo nekoliko pozitivnih i nekoliko negativnih primjera, odabiremo *najboljeg* koji će predstavljati ovaj interni čvor; korištenja mjera dobrote je objašnjena ispod.

3. Ako više uopće nema preostalih primjera, znači da smo iscrpili naučenu bazu i nije moguće klasificirati samo na temelju trenutnih primjera, pa kao jedinu mogućnost koristimo već iskorištene primjere za učenje roditeljskog čvora: vrijednost klasifikacije ovog lista se dobiva kao većinska funkcija nad (ulaznim) klasifikacijama tih primjera.

4. Inače, u situaciji smo gdje imamo neke preostale primjere, ali nemamo više atributa koji bi se testirali na ovom čvoru. To se može dogoditi kod

korištenja "šumovitih" ulaznih vektora ili zbog nedeterminističnosti domene. Možemo jedino pristupiti kao u prošloj točki: vraćamo većinsku vrijednost klasifikacije *preostalih* primjera.

Mjerenjem performansi učenja ovog algoritma, čak i bez ikakvih daljnjih poboljšanja, može se uočiti da, uz manje fluktuacije, odgovarajuća krivulja učenja asimptotski raste s brojem vektora za učenje.

3.1 Odabir najboljeg atributa za testiranje

Kao što smo spomenuli, ključan korak algoritma za učenje stabala odluke je odabir najboljeg atributa koji će se testirati na odgovarajućem unutarnjem čvoru rezultantnog stabla. Tu moramo biti relativno brzi i odabrati heuristiku koja ne zahtijeva obilazak ostatka stabla. Ovdje zato nastupa pohlepni pristup — odabirom onog atributa koji daje neku najbolju mjeru možemo biti gotovi samo u jednom prolazu kroz preostale atribute na trenutnoj razini stabla. Mjera dobrote koju pritom koristimo se temelji na *entropiji* slučajne varijable. Naime, kako radimo samo booleovsko učenje te stoga su naši ciljni atributi booleovske slučajne varijable, imamo

$$H(Cilj) = B(q) = -(q \log_2 q + (1-q)\log_2(1-q)),$$

gdje je q vjerojatnost pozitivne klasifikacije:

$$q = \frac{p}{p+n},$$

za p pozitivnih primjera i n negativnih primjera u skupu za učenje. Htjeli bismo odabrati onaj atribut koji će ostaviti što je manje posla za kasnije, tj. za kojeg je vjerojatnost da će proizvesti kratko stablo najviša. U praksi, ta je vrijednost dobro korelirana s količinom informacija koju dobivamo odabirom pojedinog atributa.

Nas dakako ne zanima globalna entropija cilja (u smislu očekivane količine informacije dobivene pri pozitivnoj klasifikaciji), već koliko informacija očekujemo ako, na trenutnoj razini stabla, odaberemo neki od dostupnih nam atributa A. Neka je $\{v_1, v_2, \ldots, v_d\}$ skup vrijednosti atributa A. Sada traženu veličinu možemo dobiti oduzimanjem očekivane entropije za A od ciljne entropije, što označavamo kao dobitak informacija (engl. *Gain*):

$$Gain(A) = B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B(\frac{p_k}{p_k+n_k}), \qquad (1)$$

gdje je d broj vrijednosti atributa A, p_k broj pozitivnih primjera među onima s $A = v_k$, a n_k slično za negativne primjere.

Pohlepni algoritam sada jednostavno treba proći kroz sve dostupne atribute A_i, izračunati $Gain(A_i)$ te odabrati onaj koji maksimizira tu veličinu. Ovaj pristup često daje jako dobre rezultate, ali postoje određene poteškoće koje ćemo sada spomenuti.

3.2 Ograničenja učenja stabala odluke i moguća poboljšanja

Jedan od glavnih problema bilo kojeg pristupa strojnom učenju je *redundantna izražajnost* (engl.*overfitting*) i on se svakako može uočiti i kod stabala učenja. Riječ je, naime, o fenomenu gdje stabla odluke mogu pokupiti razne, u stvarnosti kompletno irelevantne veze između atributa i ishoda. To je tipično za skupove podataka s velikom količinom šuma te velikim brojem (irelevantnih) atributa, a može se donekle poboljšati povećanjem broja podataka za učenje [1].

Konkretno, za naš slučaj stabala odluke, postoje razne tehnike *podrezivanja* koje rješavaju ovaj problem. To postižu eliminiranjem onih unutarnjih čvorova koji ne doprinose značajno odluci, počevši od listova izgrađenog stabla i penjajući se uz stablo. Pritom se ograničavamo na promatranje samo onih unutarnjih čvorova čija su sva djeca listovi. Radi određivanja irelevantnosti atributa, opet koristimo informacijski pristup; naime, očekujemo da bi odabir kompletno irelevantnog atributa ostavio udio preostalih pozitivnih atributa otprilike jednakim početnom $p/(p+n)$. Po (1), vidimo da će tada dobitak informacija za ovaj atribut biti jako blizu nuli. No pitanje je: koji prag dobitka uzeti za podrezivanje?

Odgovor leži u primjeni testova značajnosti — prvo pretpostavimo da dani atribut zaista nema nikakvu regularnost i stoga nam ne može dati nimalo informacija (tzv. nul-hipoteza). Zatim utvrdimo koliko taj atribut odstupa od te idealne irelevantnosti i prema tome zaključimo želimo li ga odbaciti ili ne. Ovdje se tipično koristi neka dobro poznata statistička distribucija koja opisuje očekivana odstupanja te se uzima mali prag ispod kojeg se smatra da postoje značajne regularnosti u danom atributu, te ga onda ne odbacujemo. Inače, on je praktički "šum" i možemo ga slobodno odbaciti. [1]

Još jedan od problema kod učenja stabala odluke koji valja spomenuti je baratanje s *numeričkim* atributima. Naime, do sada smo uglavnom pretpostavljali da će stablo obrađivati kategoričke atribute, kod kojih ne očekujemo velik raspon vrijednosti; s druge strane, ako nam se na ulazu pojave atributi s domenom u proizvoljnom (recimo realnom) skupu brojeva, onda više ne možemo očekivati da će postupak učenja ikada uočiti sve vrijednosti takve domene. Također, naivni pristup učenju ovakvih atributa može dovesti do

jako visokoh i nerazumljivih stabala. Rješenje je u particioniranju domene svih numeričkih atributa na način da odabrane točke podjele čine diskretiziranu domenu. Pritom također želimo odabir točaka podjele koji maksimizira očekivan dobitak informacija (naspram svih mogućih drugih odabira, iako najčešće ovo samo aproksimiramo pohlepnom, jednoprolaznom heuristikom).

4 Opis softverskog rješenja

4.1 Korištenje i struktura programa

Dio ovog rada je i Java program koji implementira osnovni algoritam učenja stabala odluke kao i sva iznad spomenuta poboljšanja. Osnovni pristup učenju je temeljen na algoritmu iz [1], s dodatkom koji omogućuje istovremeni rad i s kategoričkim i s numeričkim atributima.

Program preko standardnog ulaza prima naziv jedne CSV datoteke s prvim retkom jednakim listi naziva svih atributa te preostalim linijama s odgovarajućima vektorima vrijednosti. Klasifikacijski atribut mora biti *zadnji* u CSV-u. Uz to, program očekuje još onoliko dodatnih ulaza koliko ima i atributa (jasno, bez klasifikacije!) radi primanja novog primjera za klasifikaciju. Taj primjer će se provesti kroz izgrađeno stablo odluke nakon njegove izgradnje (i opcionalnog podrezivanja — trenutno je standardno uključeno) i korisniku će se na standardni izlaz objaviti izračunata predikcija klasifikacije (+ ili −). Podaci za ulazni primjer moraju biti uneseni u redoslijedu koji točno prati onaj iz ulazne CSV datoteke te se ne smije izostaviti niti jedan atribut. Na kraju, korisnik također u radnom direktoriju dobiva i `PNG` sliku generiranog stabla odluke.

Program je pisan u objektno-orijentiranom stilu. Glavna klasa za interakciju s korisnikom je u `DecisionTrees.java`, dok klasa u `DepthFirstTree.java` implementira sve funckionalnosti vezane za učenje stabala odluke (na način koji liči DFS-u, stoga ime). Ta klasa ovisi o klasi u `AttributeSet.java` koja implementira apstrakciju skupa svih atributa i njihovih vrijednosti za danu instancu problema učenja. Takva apstrakcija je potrebna zato što nam pri diskretizaciji i možda nekim dodatnim post-procesima nad ulaznim podacima treba pristup svim atributima kao i rezultatima njihove klasifikacije na ulazu.

Program koristi Apache Commons CSV i Math biblioteke radi fleksibilnog CSV parsiranja i računanja veličina vezanih za statističke distribucije, kao i Graphviz-java za vizualizaciju stabala. Popratna programska dokumentacija je u priloženom `javadoc` direktoriju, kao i u komentarima prožetim kroz kôd.

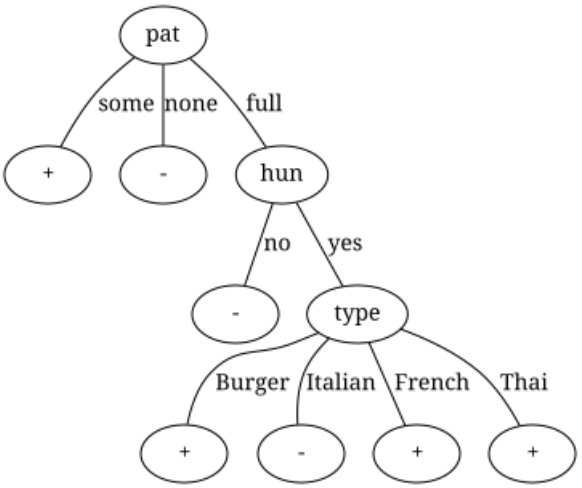
4.2 Implementirane funckionalnosti

Pri particioniranju numeričkih atributa se koristi *segmentiranje pod nadzorom* (engl. *supervised binning*), pristup ukratko opisan u [1] i prikazan u [3]. Ideja je da možemo u samo jednom prolazu kroz sve uočene numeričke vrijednosti danog atributa napraviti kompletnu "optimalnu" (u pohlepnom smislu) particiju ukoliko prije početka particioniranja sortiramo sve te vrijednosti. Tada algoritam postaje jednostavan rekurzivan, podijeli-pa-vladaj zadatak pri čemu samo treba pripaziti na baratanje s duplikatima, što je ipak olakšano zbog sortiranosti. Pohlepnost dolazi do izražaja na praktički jednak način kao kod računa (1) iznad, samo što se sada maksimizira dobitak pri odabiru točke podjele raspona vrijednosti, a ne odabira cijelog atributa. Točka podjele mora ležati između dviju susjednih ali različitih klasifikacija, dakle pri prijelazu s + na − ili obratno. Ovaj algoritam je implementiran u metodi `bin` u `AttributeSet.java`.

Objasnimo još implementaciju podrezivanja izgrađenog stabla, koja koristi χ^2 distribuciju. Pretpostavimo da je na trenutnom čvoru stabla dostupan uzorak od $v = n+p$ primjera. Mi želimo izračunati očekivano odstupanje (pod nul-hipotezom) od stvarno izmjerenog. Dakle, za atribut ovog čvora moramo izračunati očekivani broj pozitivnih primjera $\hat{p}_k = p\ \frac{p_k+n_k}{p+n}$ i očekivani broj negativnih primjera $\hat{n}_k = n\ \frac{p_k+n_k}{p+n}$ za svih d različitih vrijednosti ovog atributa ($k = 1, 2, \ldots, d$). Zatim ukupno odstupanje možemo dobiti kao

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

jer je pod nul-hipotezom očekivana vrijednost te veličine distribuirana upravo po χ^2 distribuciji s $v - 1$ stupnjeva slobode (suma pozitivnih i negativnih primjera na ovoj razini za dani atribut jest upravo broj svih relevantnih primjera). Kako mi imamo zadanu gornju među p-vrijednosti tj. vjerojatnosti da nul-hipoteza vrijedi za dano odstupanje, a imamo izračunato ukupno odstupanje Δ, sada moramo dobiti *donju* među odgovarajućih vrijednosti χ^2 distribucije kako bismo vidjeli je li naša Δ prihvatljiva; ako je premala, p-vrijednost je prevelika da bismo bili sigurni da nul-hipoteza ne vrijedi pa odbacujemo atribut. Trenutni prag p-vrijednosti je u programu postavljen na 0.05, što je uobičajena vrijednost [1]. Spomenutu donju među možemo dobiti računanjem inverzne CDF odnosno kvantila χ^2 distribucije za kumulativnu vjerojatnost $1 - p$, što radimo pomoću Apache Common Math biblioteke.



Slika 1: Stablo iz primjera restorana

5 Primjeri

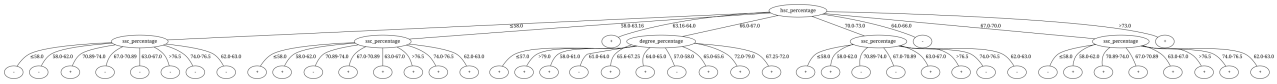
5.1 Odabir restorana

U [1] je dana mala tablica (12 primjera) s podacima koji odgovaraju nečijem odabiru restorana ovisno o raznim okolnostima, poput razine gladi, tipu hrane koja se nudi, postoji li prethodna rezervacija itd. Primjenom izrađenog programa (koji na kraju izvodi i podrezivanje stabla) na taj skup podataka dobivamo stablo sa slike 5.1.

Klasifikacija novog primjera s ulazima (alternativa?,bar?,vikend?,gladan?,gostiju?,cijena,kiša?, čekanje) =($\top$, $\bot$, $\bot$, $\top$, puno,srednja, $\top$, $\bot$, talijanski , $10-30$) daje negativnu predikciju.

5.2 Zapošljavanje svježih diplomanata

Sljedeći primjer koristi bazu podataka za učenje sa stranice Kaggle [2] kako bi predvidio hoće li se diplomant na ulazu (opisan u terminima predmeta studija na različitim razinama obrazovanja, ocjena tijekom cijelog školovanja, rezultata standardiziranih testova i sl.) moći odmah zaposliti tj. biti primljen od strane neke firme ili ne. Kako je originalan skup podataka poprilično velik i raznolik, rezultirajuće stablo je preveliko za prikazati ovdje, pa na slici 5.2 prikazujemo jedno manje stablo dobiveno na podskupu ovih primjera, nakon obrezivanja.



Slika 2: Stablo iz primjera zapošljavanja

6 Zaključak

Ovaj rad je ukratko objasnio kako se može učiti sa stablima odluke, njihov značaj u problemima booleovske klasifikacije te koje su im prednosti i mane. Vidjeli smo neka moguća poboljšanja osnovnog algoritma za učenje te proučili kako se može raditi s numeričkim atributima povrh standardnih kategoričkih. Također smo implementirali i ukratko objasnili funkcioniranje i implementaciju popratnog programskog rješenja te pokazali kako radi paru primjera.

Daljnje primjene stabala odluke uključuju i viševrijednosnu klasifikaciju, tj. probleme klasifikacije s većim domenama od booleovske, što u limitu dovodi do tzv. regresijskih stabala — problema učenja gdje su ishodi neprekidne varijable. Tada više nije riječ o klasifikacijskom, već o regresijskom problemu jer je nužno da listovi rade linearnu regresiju nad podskupom numeričkim atributa. To onda zahtijeva i složenije algoritme koji moraju moći odlučiti u kojem trenutku izgradnje stabla stati i postaviti regresijski list na trenutnu granu.

Literatura

[1] Stuart Russell i Peter Norvig. *Artificial intelligence: A modern approach, global edition*. 3. izdanje. London, England: Pearson Education, travanj 2016.

[2] Ahsan Raza. *Job Placement Dataset — Kaggle*. URL: `https://www.kaggle.com/datasets/ahsan81/job-placement-dataset`.

[3] Saed Sayad. *Supervised Binning*. Pristupljeno = 2023-02-26. URL: `https://www.saedsayad.com/supervised_binning.htm`.