

TDK DOLGOZAT

Szerző:

Bán Péter

Miskolci Egyetem



GÉPÉSZMÉRNÖKI- ÉS INFORMATIKAI KAR
INFORMATIKAI INTÉZET

NEM BINÁRIS REGRESSZIÓS DÖNTÉSI FÁK VIZSGÁLATA

KÉSZÍTETTE:

Bán Péter

Másod éves Mérnökinformaticus szakos hallgató

KONZULENS:

Prof. Dr. Kovács László

egyetemi tanár

Lezárás dátuma:

Miskolc, 2024.05.23

Bevezetés

Napjaink egyik legismertebb informatikai témaköre a mesterséges intelligencia. A témakör pontos meghatározása viszont nem egy egyszerű feladat. Már az se egyértelmű, hogy mit tekintünk intelligenciának, hiszen még erről is vita folyik. Alan Turing kidolgozott egy tesztet [35] arra, hogy megállapítsuk hogy a mesterséges intelligencia valóban intelligens-e. Ennek során, ha egy ember nem tudja megkülönböztetni a "gépet" az embertől egy párbeszédet követően, akkor a gépet intelligensnek tekinthetjük. Ezzel a tesztel az a probléma, hogy ez a teszt is a valódi intelligenciának csak egy szeletét fedi le. Éppen ezért hasznosabb, ha úgy tekintünk a mesterséges intelligenciára, mint algoritmusok, módszerek, eszközök együttese, amely képes egy adathalmaz segítségével tanulni és olyan problémákat megoldani, amihez emberi gondolkodás szükséges. [1]

A mesterséges intelligenciák alkalmazása elősegítik és felgyorsítják bizonyos problémák megoldását. A vizsgálatok tárgya a regresszió és az osztályozás mely többféle algoritmust ölel fel. Az osztályozás során a meglévő attribútumok alapján osztályokba soroljuk a kérdéses elemeket. Regresszió során pedig az attribútumok alapján egy valós érték meghatározása a cél. Ezekre a feladatokra többféle eszközt is kidolgoztak, köztük a döntési fákat. [5]

A döntési fák már több elemzés alkalmával bizonyították hasznosságukat a valós problémák megoldása során.

A döntési fa módszert sikerrel alkalmazták arra, hogy bizonyos területekre mennyire jellemző az árvíz előfordulása egy év folyamán, és hogy melyek azok a tényezők, amelyekből következtethetünk az árvíz előfordulására. A CART algoritmust alkalmazták, amely 83% pontossággal teljesített és ennek segítségével még egy térképet is létre hoztak melyen jól kivehető, hogy mely területek érintettek nagyobb eséllyel árvizekben.[2]

Az energia szektorban azon belül is a megújuló energia kapcsán is nagy hasznát vették a döntési fáknak. Annak érdekében, hogy lerövidítsék azt az időt, amivel meghatározzák egy szélerőmű elhelyezésének ideális helyét (hosszadalmas méréssel) döntési fákat használva állították elő az ehhez szükséges információkat. Az eredmények végül jelentősen megközelítették a valós adatokat 95%-os pontossággal [3].

A döntési fákat alapvetően osztályozásra találták ki, és az irodalomban is döntően erre látunk példákat, viszont megfelelő adaptálással képesek regressziós feladatot is ellátni. Két széles körben

ismert és használt döntési fa algoritmus az ID3, valamint a CART. Az előbbi osztályozásra még az utóbbi mind a kettőre használható. Az ID3 algoritmust 1986-ban publikálta Quinlan, J. Ross [4]. Ő kiemelte, hogy a mesterséges intelligenciának több területe létezik. Vannak olyan rendszerek, amelyek a saját teljesítményüket alapul véve belső paramétereik állításával(finomhangolásával) próbálnak egyre jobb eredményeket elérni. Egy másik megközelítés pedig amikor a tanulás folyamán strukturált tudásra tesz szert. Viszont az akkori módszerekkel ezeknek a rendszereknek megvoltak a határai mivel több száz vagy akár több ezer szabály megtanítására lenne szükség. [4]. Az ID3 egy egyszerű tudásalapú rendszer, mely könnyen tanítható. Ez osztályozásra lett kitalálva, és azóta már létezik több tovább fejlesztett változata is, ebből az egyik legismertebb a C4.5. Erre azért volt szükség, mert az ID3 érzékeny volt afelé az attribútumok felé, amelyek relatívan sok célértékkel rendelkeznek. [4] A való életben sok adat, mint például a hőmérséklet vagy a tömeg folytonos értékek így az osztályozás nem egy megfelelő módszer. Ilyenkor használhatunk egy regressziós fát.

Felügyelt és felügyelet nélküli tanítás

A felügyelt tanítás során az algoritmust egy olyan T tanító halmaz segítségével tanítjuk, amelyben az objektumok két fő részre bonthatóak. A függő és független változókra. A független változók attribútumokból állnak össze a függő változók pedig azok az értékek, amelyeket meg szeretnénk állapítani a függő változók alapján. Ez osztályozás esetén diszkrét értékek, regressziós esetén pedig valóságok. A független változók alapján az algoritmus megpróbál következtetni a hozzá tartozó függő változókra. A pontosság alapján pedig a megfelelő változtatásokat hatjuk végre a modellben annak érdekében, hogy a jövőben pontosabb eredményt adjon. Ilyen módon tanulnak például a neurális hálók. Ez a módszer további két részre bontható. Osztályozásra és regresszióra. Osztályozás során egy előre definiált osztályokba sorolást hajtunk végre, még regresszió során pedig valós értékeket szeretnénk meghatározni az attribútumok alapján minél nagyobb pontossággal.[5]

A felügyelet nélküli tanítás során egy T' adathalmazt alkalmazunk. A T' egy felcímkézetlen adathalmaz melyben nincsenek célértékek. A meglévő adatokból, különböző módszerekkel egy ember számára is értelmezhető és hasznos adatokat próbál kinyerni a modellünk. Egy ilyen módszer például a klaszterezés mely során az adatokat különböző csoportokba soroljuk és így elkülönítjük őket egymástól. [5]

Döntési fák

Az első döntési fa a szakirodalomban még 1963-ban jelent meg, az Automatic Interaction Detection (AID) [6]. Ez a döntési fa regressziós feladatok megoldására lett kitalálva. A fa gyökér csomóponttól kezdődően minden csomópontban binárisan ketté osztja az adathalmazt. Az impurity az a mérőszám, amely meghatározza hogy egy felosztás mennyire optimális. Erre az átlagtól való négyzetes eltérést használták. Ez alapján hajtotta végre a felosztást az algoritmus, ott, ahol a legkisebb volt ez az eltérés. Amennyiben elér egy bizonyos előre meghatározott heterogenitást, akkor nem folytatja a felosztásokat. A jóslat érték pedig a végső csomópontban lévő célértékek átlaga.[7]

Egy későbbi jelentős továbbfejlesztés a CART algoritmus amelyet 1984-ben publikáltak, mely a regresszió mellett már osztályozást is képes volt elvégezni.[8]. Hogy milyen feladatok elvégzésére lett kitalálva azt a neve is tükrözi, amely a Classification and Regression tree rövidítése. Egy nagy újítása az algoritmusnak, hogy ahelyett, hogy valamilyen szabály alapján leállna a fa épülése, engedi hogy egy nagy méretű fát kiépítsen és utána vissza metszi. Ez megoldja az alul illeszkedés és felül illeszkedés problémát, amivel az AID küzdött. Cserébe viszont több számítási kapacitást igényel. Emelet képes kezelni a hiányzó adatokat is.[7]

Az ID3(Quinlan, J. Ross 1986) egy osztályozásra fejlesztett fa mely nem képes regressziós feladatot megoldani. [7]. Ez egy jól működő algoritmus viszont vannak hibái. Amennyiben egy attribútumhoz aránylag sok érték tartozik, afelé elfogult lesz, valamint a zajos adathalmazt sem képes megfelelően kezelni [9]. A C4.5(Quinlan, J. Ross 1993) [9] az ID3 továbbfejlesztése, amely az előbb felsorolt hibákat javítja valamint lehetővé teszi a regressziós feladatok megoldását is. Hogy ne legyen elfogult egy új módszert használ. Egy csomópont impurity mérésére az Information gain helyett a Gain ratio-t használja, amely képes az adathalmazban történő elfogultságokat kiszűrni.[9]

Ezek a módszerek alapjait képzik olyan más kutatásoknak, amelyek a jelenünkre is kihatással vannak és egy átlagembert is érinthet, vagy a szakemberek munkáját segítik elő.

A kutatás és fejlesztés területén is alkalmaztak döntési fákat annak érdekében, hogy elősegítsék a mérnökök fejlesztői munkáját. Olyan rendszereket terveztek amelyek ezt különböző módszerek segítségével valósítják meg mint a KNN, SVM valamint döntési fa és regressziós fa. Ezen kutatás

során két adathalmazon vizsgálták a különböző módszereket, algoritmusokat. Az egyik esetben motorok még a másik esetben kompresszorok adatait használták és dolgozták fel. A regressziós fa az esetek többségében 90% körüli pontosságot ért el. A stator magasságának megállapításakor a Random forest 91%-os pontosságot ért el még az LS BOOST csak 84%-ot az SVM linear pedig 86-ot. A rotor magasságának megállapítása esetén a regressziós fa 89%-os pontossággal rendelkezett még a KNN csak 84%-al. [10]

Az önvezetés technológia rohamosan fejlődik és egyre több autóban elérhető valamilyen korlátozással, mivel még vannak korlátai a technológiának. A kamerák „látási” viszonyait nagyban befolyásolja az időjárás, éppen ezért, hogy megállapítsák az út állapotát radar szenzorok adatait használták fel egy kutatás során és egy CART döntési fát építettek ki. Az esetek megközelítőleg 95% -ban pontos eredményt adott [11].

A regressziós fa továbbá hasznosnak bizonyult az élelmiszeriparban is. Azt szerették volna megállapítani, hogy hogyan őrizték meg az áfonya minőségét addig még el nem jut a fogyasztóig. Ebben többek között fontos szerepet játszik a hőmérséklet és a páratartalom. Egy BRT modellt használtak, amely nagy pontossággal meg tudta jósolni az áfonya súly csökkenését amely alapján óvintézkedéseket lehet bevezetni [12].

ID3

A T tanító halmaz attribútumokból és célértékekből épül fel. A az attribútumok halmaza, valamint C a célváltozók halmaza. A_j egy attribútumhoz tartozó értéke halmaza.

$$x_{i,j} \in \text{dom}(A_j)$$

Melyben $x_{i,j}$, A_j i -ik eleme.

A döntési fa egy $G = (V, E)$ gráf ahol V a csúcspontok halmaza, E pedig az élek halmaza. A $v \in V$ csúcspont egy feltételt jelöl, amely valamely attribútumra vonatkozik, és ez alapján felosztjuk az T adathalmazunkat. Az $e \in E$ él pedig a feltétel kiértékelését követő eredmény. $\alpha(v)$ a döntések kimenetelének halmaza, a $C(v)$ pedig gyerekek halmaza. [13]

A döntési fa felőlről lefelé építkezik. A gyökércsomópontból kiindulva meghatározunk egy feltételt ez meghatározza az elágazásokat, majd azt követően felosztjuk az adathalmazt. Ezáltal eljuthatunk

a következő szinten található csomópontokig. Ezt rekurzívan addig folytatjuk egy ágon még el nem érünk egy levélig. [14].

Maga a feltétel meghatározása, a felosztás, valamint, hogy mikor érünk el egy levelet implementáció függő. Az ID3 esetén addig folytatjuk a felosztást még fel nem használtunk minden attribútumot legfeljebb egyszer, hogy ne ragadjunk bele egy végtelen ciklusba, vagy amíg homogén nem lesz a célváltozók halmaza. Azt, hogy melyik attribútum alapján osztjuk fel az adathalmazt azt az information gain fogja meghatározni. Amelyik felosztásnál a legnagyobb volt az information gain, aszerint az attribútum szerint osztjuk fel az adathalmazt.[15]

Ahhoz, hogy megállapítsuk az information gaint szükségünk van az entrópiára. Az entrópiával meghatározhatjuk egy csomópont heterogenitását.

Az entrópiát a következő képlettel számoljuk:

$$En = - \sum_{i=1}^c p_i \times \log_2 p_i$$

Ahol En az entrópia, c a cél kategóriák száma, p_i pedig az i -edik kategória előfordulási valószínűsége. Amennyiben az entrópia 0 az azt jelenti, hogy teljesen homogén a célváltozók halmaza, egyhez közeledve pedig csökken a homogenitás (1-nél nagyobb nem lehet az entrópia).[15]

Például ha két kategóriánk van a és b valamint a halmazunk úgy nézz ki hogy $\{a,a,b,b\}$ akkor az entrópiát a következő képen számoljuk ki:

$$En = - \left(\frac{1}{2} \times \log_2 \frac{1}{2} + \frac{1}{2} \times \log_2 \frac{1}{2} \right) = 1$$

Ez messze van a homogenitástól. Azonban, ha például a halmazunk úgy nézz ki hogy $\{a,a\}$ akkor az entrópiát a következő képen számoljuk ki:

$$En = - (1 \times \log_2 1) = 0$$

Szóval teljesen homogén az adathalmazunk.

Az information gain az entrópia várható csökkenése az adott attribútum felosztás során [14]. Ennek segítségével pedig meg tudjuk határozni, hogy melyik attribútum szerint érdemes felosztani az

adathalmazt. Ilyenkor kiszámítjuk a szülő csomópont entrópiáját és kivonjuk belőle a gyerek csomópontok súlyozott entrópiáját [14][15].

Az information gaint a következő képlettel számoljuk:

$$I_g = E_p - \frac{\sum_{i=1}^k (n_i \times E_i)}{N}$$

Ahol E_p az a szülő csomópont célkategóriáinak az entrópiája, n_i az i-edik gyerek csomópont kategóriáinak számossága, az $entropy_i$ pedig az i-edik gyerek csomópont kategóriáinak az entrópiája. Az N pedig a bejövő objektumok számossága.[15]

Amikor a fát felépítjük akkor rekurzívan felosztjuk az adathalmazt. Minden attribútummal végrehajtjuk a felosztást majd kiszámítjuk a létrejött részhalmazok entrópiáját. Ezt követően meghatározzuk az I_g -t és az összes felosztást követően azt választjuk amelyik esetén a legnagyobb volt az I_g . [15]

Végül miután felhasználtunk minden attribútumot, vagy teljesen homogén a célváltozók halmaza akkor az adott ágon befejezzük a fa építését. Az itt található kategóriák reprezentálják a döntést. [14][15]. Amennyiben meg szeretnénk hozni egy döntést akkor az attribútumokon végrehajtjuk a korábban meghatározott tesztekéket míg el nem jutunk egy döntésig. Például, ha azt szeretném eldönteni, hogy elmenjek-e kerékpározni, amelyet 0-val jelölök, ha nem és 1-el ha igen, azt egy attribútum fogja meghatározni, az pedig az eső. Egy tesztet követően erre két lehetséges válasz lehet igen és nem. Amennyiben esik az eső az igen feltételt teljesítő ágon haladok tovább itt viszont csak olyan értékek vannak, ahol a célváltozók a 0 értéket veszik fel. Ellenkező esetben pedig csak olyan célváltozóim lehetnek, amelyek értéke 1.

C4.5

A C4.5 az ID3 továbbfejlesztése, amely annak hiányosságait, hibáit hivatott orvosolni. A legfőbb probléma, hogy amennyiben egy adott attribútumhoz relatívan sok célérték tartozik akkor túlságosan nagy szerepet fog neki tulajdonítani. Ennek következtében könnyedén előfordulhat, hogy nem a legoptimálisabb felosztást fogja elvégezni [7]. A való életben több ilyen probléma is felmerült [7] és ennek következtében jött létre a C4.5 megoldása, amely egy új kritériumot fog használni egy megfelelő felosztás megtalálásához. Egy ilyen probléma az volt, hogy egy

egészségügyi feladat során a 'beteg életkora' attribútumot választotta. Viszont a szakértő szerint ez nem bírt olyan jelentőséggel, mint más attribútumok. Az ID3 algoritmus azért választotta a 'beteg életkora' attribútumot mert relatívan sok érték tartozott hozzá [7].

Hogy jobban megértsük a problémát képzeljük el, hogy van egy olyan attribútumunk, amelynek minden értéke különböző és véletlenszerű. Ennek az attribútumnak maximális lenne az I_g értéke így ezt választaná a felosztáshoz. Ez valójában semmilyen módon nem hasznos számunkra mivel az attribútum minden értéke véletlenszerű volt.[7] A C4.5 algoritmus annak érdekében, hogy ezt kiküszöbölje egy új módszert alkalmaz. Az I_g helyett a Gain ratio-t fogja használni. Ahhoz, hogy alkalmazni tudjuk a Gain ratio-t szükségünk van egy másik függvényre. Ez pedig a Split info. [7]

A Split infot-t a következő képlettel számítjuk ki:

$$S_i = - \sum_i^n \frac{i}{p} \times \log_2 \frac{i}{p}$$

Ahol az $\frac{i}{p}$ az i-edik tesztben a célkategóriák aránya. Az S_i -t felhasználva kiküszöbölhetjük az ID3 elfogultságát. Az I_g helyett a Gain ratio-t fogjuk használni. [7][16]

A Gain ratio-t következő képlettel számoljuk ki:

$$G_r = \frac{I_g}{S_i}$$

G_r feladata, hogy normalizálja az I_g -t így egy valósabb képet fog alkotni. Ezáltal elkerülhető az algoritmus elfogultsága. [7][16]

A C4.5 továbbá képes kezelni valós értékekekel rendelkező attribútumokat is [17]. Amennyiben valós értékeket szeretnénk kezelni azt úgy tehetjük meg hogy csoportokba tesszük őket. Először is növekvő sorrendbe kell rendezni az attribútum értékeket $A_1, A_2, \dots, A_u$, majd azt követően besorolni őket csoportokba, amelyeket jelöljön $C_1, C_2, \dots, C_v$. Például, ha $A_i > A_m$ és $A_i \leq A_n$ akkor a C_k csoportba soroljuk, ahol A_m, A_n C_k alsó, illetve felső határa. [17]

Regresszió

Regressziós feladat során folytonos vagy matematikailag értelmezve valós értékek meghatározása a cél. Regressziós analízis során rendelkezésünkre állnak a függő és független változók. A független változókat felhasználva próbálja a lehető legnagyobb közelítéssel megjósolni a célértékeket. A feladat, hogy a modellben egy függvény család legjobban illeszkedő tagját használjuk. Erre különböző módszerek léteznek ilyen például a lineáris regresszió és a polynomiális regresszió. A polynomiális regresszió visszavezethető lineáris regresszióra így az valójában a lineáris regresszió egy speciális esete.[5][18][19] [24][25]

Regressziós fa

A regressziós fa, mint az a nevéből is adódik regressziós feladatokat képes megoldani. Ezt úgy teszi meg hogy a tanító halmazt minden egyes mélységi szinten kettő részre bontja. Ahhoz, hogy meghatározza, hogy mely érték alapján végezzük el a felosztást azt tesszük, hogy egy adott attribútum alapján sorba rendezzük az adathalmazunkat majd ezt követően vesszük két szám középértékét és ennél az értéknél végrehajtjuk a felbontást.[15]

$$Z_i = \frac{x_i + x_{i+1}}{2}$$

Ahol az Z a határpontok halmazát jelöli az x_i az i -ik érték az A_j halmazban az x_{i+1} pedig az x_i után következő attribútumot jelöli.

Ezt addig folytatjuk még végig nem iteráltunk az A_j összes értékén. Arra, hogy mely Z_i értéket válasszuk ki végül azt úgy határozzuk meg, hogy a két keletkező halmaz szórását összeadjuk, és amely felosztásnál a legkisebb ez az érték ott fogjuk ketté bontani az adathalmazt.[15][21]

Ezzel a módszerrel rekurzívan addig építjük a fát még el nem érünk egy bizonyos mélységet, amelyet előre meghatározunk vagy még minden attribútumot fel nem használtunk egy előre meghatározott alkalommal. [15] Mivel itt nem fordulhat elő, hogy egy végtelen ciklusba keveredünk ezért egy attribútumot többször is felhasználhatunk. Azért nem keveredhetünk végtelen ciklusba mert ha egy attribútum alapján elég sokáig végzünk felosztásokat akkor minden

elem külön Csomópontba kerül és véget ér a fa építése. Ez nem optimális, mivel rengetek felesleges felosztást végez, továbbá túlilleszkedik ekkor és nem fog általánosítani megfelelően

Lineáris regresszió

A lineáris regresszió esetén egy olyan függvényt, lineáris modellt hozunk létre, amely egy attribútum ismeretében képes következtetni a célértékekre. A lineáris modell használatának célja, hogy kihasználjuk az adathalmazunkban előforduló lineáris összefüggéseket. [19][22]

Az egyváltozós lineáris modell a következő képen írható fel:

$$\hat{y} = ax + b$$

Ahol $\hat{y}$ a függvény érték, az a a függvény meredekségét adja meg, az x a bemeneti érték, és b pedig az eltolás mértékét jelenti.

Többváltozós lineáris regresszió esetén pedig:

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 + \dots + b$$

Ahol $\hat{y}$ a függvény érték, az θ_i az x_i a bemeneti érték együtthatója, és b pedig az eltolás mértékét jelenti.[23]

Polinomiális regresszió

A polinomiális regresszió esetén egy olyan függvényt, polinomiális modellt hozunk létre, amely attribútumok ismeretében következtet a célértékekre. A polinomiális regresszió a lineáris regresszió egy különleges esete. Például, ha az x^2 -et nem úgy értelmezzük, mint az x négyzetre emelt értéke hanem mint egy teljesen független változó akkor visszavezettük egy többváltozós lineáris regresszióra. Az együtthatók és az eltolás mértéke továbbra is lineáris [24][25]

A polinomiális modell a következő képpen írható fel:

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1 + \dots + b$$

Ahol $\hat{y}$ a függvény érték, az θ_i az x_i a bemeneti érték együtthatója, és b pedig az eltolás mértékét jelenti.

A lineáris modellt úgy alkotjuk meg hogy a lehető legjobban illeszkedjen az adathalmazunkra. Ahhoz, hogy ezt elérjük egy hiba függvényt használhatunk. Több ilyen hiba függvény is létezik, ezek közül gyakran használt a Mean Squared Error.[25]

A Mean Squared Error-t a következő képpen számítjuk ki:

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

Ahol az N az adathalmaz számossága, $\hat{y}_i$ az i-edik eset jósolt értéke y_i pedig a valós eredmény.

A cél az, hogy az MSE minimális legyen, mert akkor található a legkevesebb eltérés, vagyis ekkor illeszkedik a legjobban a függvényünk az adathalmazhoz. Ez akkor lesz minimális, ha a függvény deriváltja 0, mivel ott a függvénynek szélsőértéke van. Amikor elértük ezt a minimális értéket akkor az ott használt függvényparamétereket fogjuk megtartani és használni a jövőben.[25]

Hogy eljussunk ezekhez a paraméterekhez ahhoz a Gradient Descent módszert alkalmazhatjuk.

Annak érdekében, hogy elérjük az MSE függvény minimumát egy iteratív megoldást használhatunk. A modell létrehozásakor tetszőleges paramétereket használunk mivel nem tudjuk melyek lesznek azok, amelyek a legjobb eredményt fogják adni. Ezt követően pedig a megfelelő számításokkal elkezdünk közelíteni az optimális paraméterekhez.[25][26]

Egy egyváltozós lineáris modell esetén:

$$a = a - \sum_{i=1}^n \frac{\partial MSE}{\partial a} \times L$$
$$b = b - \sum_{i=1}^n \frac{\partial MSE}{\partial b} \times L$$

Ahol az L a learning rate, amely a lépések méretét határozza meg a MSE függvény adott pontjánál található meredekséggel együtt, amellyel a függvény minimumát közelítjük. A folyamat során iterációnként a teljes adathalmazra nézve kiszámítjuk a parciális deriváltakat majd azt követően azt megszorozzuk L-el amellyel megkapjuk a lépést. Végül pedig kivonjuk a korábbi paraméterünkből és ezt ismételjük míg el nem értünk egy maximális iterációs számot vagy már túl kicsi a lépés mérete. [25][26]

A regresszió egy fontos problémája az alul és felül illeszkedés. Ez azt jelenti, hogy a modellünk nem volt képes megfelelően általánosítani az adathalmaz alapján. Ez következhet a modell hiányosságából, nem megfelelő paramétereiből vagy nem megfelelő adathalmazból.[5]

Függvényosztály alapú regression tree.

A függvényosztály alapú regressziós fa abban különbözik más regressziós fáktól, hogy a csomópontok és a levelek nem csak értékekből állnak, hanem többváltozós lineáris modellekből. Mint azt J. R. QUINLAN kutatása is mutatja az M5 algoritmus kapcsán, egy nagy előnye a függvényosztály alapú regressziós fának, hogy egy pontosabb döntési fát építhetünk fel. Mindemellett még korábban nem látott eseteket is jól tud kezelni mivel a lineáris modell kiterjeszthető eddig nem látott értékekre. [27]

Hierarchical Agglomerative Clustering(HAC)

A HAC egy felügyeletlen algoritmus. Ez egy alulról felfelé építkező módszer. Eleinte minden egyes adatpont egy klasztert alkot. Ezt követően kiszámítjuk a klaszterek közötti távolságokat és a két legközelebbit egy klaszterbe egyesítjük. Mindezt addig folytatjuk amíg el nem érünk egy bizonyos kritériumot például egy előre meghatározott klaszter számot. Ezt a módszert nem csak arra használhatjuk, hogy a végén létrejött klaszterekből információt nyerjünk ki, hanem dendrogramon való ábrázolással akár végig kövessük hogy hogyan alakulnak ki a klaszterek amelyek szintén értékes információval szolgálhatnak. A klaszterek közötti távolság kiszámítására különböző módszereket lehet használni. Ilyenek a ward, complete, single és average.[28]

Meglévő implementációk

Manapság számos implementáció található az interneten. Az általam bemutatott implementációk a scikit-learn python könyvtárban találhatóak meg.

Döntési fa

Amennyiben a Döntési fát osztályozásra szeretnénk használni, igénybe vehetjük DecisionTreeClassifier osztályt. Bemenetként két tömböt vár. Az egyik az X, amely egy kétdimenziós tömb és a méretei az n_samples ami a példák száma valamint az n_features amely pedig tulajdonságok száma, ezekben találhatóak az attribútumok, valamint a y tömböt amiben pedig az attribútumokhoz tartozó osztályok találhatóak.[29][30]

Miután létre hoztunk egy példányt a `DecisionTreeClassifier` osztályból azt követően a `fit()` metóduson keresztül felépíthetünk egy fát. Végül a `predict()` metódust használva az átadott attribútumok szerint vissza adja azt az osztályt aminek a legnagyobb a valószínűsége az adott ágon. Alternatívaként használhatjuk a `predict_proba()` metódust amely az egyes osztályok valószínűségét adja meg az adott ágon.[29][30]

Regressziós feladat esetén használhatjuk `DecisionTreeRegressor` osztályt. Hasonlóan az osztályozáshoz itt is egy `X` és `y` tömböt kell át adni, annyi különbséggel, hogy az `y` tömbben egész értékek helyett lebegőpontos értékeket vár. [29][30]

A `fit()` és `predict()` metódus a fentebb leírtaknak megfelelően használhatóak. [29][30]

HAC

HAC esetén használhatjuk a `AgglomerativeClustering` osztályt. Amikor létrehozuk a példányt megadhatunk többféle paramétert.

Az `n_clusters` megadja, hogy mennyi az a klaszter szám amelyet követően ne folytassa tovább az egyesítéseket.

A `linkage`-el pedig különböző módszereket adhatunk meg arra, hogy hogyan számítsuk ki két klaszter között a távolságot. Ennek kiszámítása többféle módon történhet. A könyvtárban a következő módszerek találhatóak meg: `ward`, `complete`, `average` és `single`. [28][31]

A `ward` esetén azon két klasztert fogja egyesíteni amelyek variációja a legkevesbé tér el. Az `average` esetén a távolság a pontok minden egyes másik klaszterben lévő pontok távolságának átlaga. A `complete` a két klaszter közötti legnagyobb távolságú pontokat használja. A `single` pedig a két klaszter közötti legközelebbi pontokat veszi alapul.[28][31]

Linear regression/polynomial regression

Lineáris regresszió esetén használhatjuk a `LinearRegression` osztályt. Az osztály egy lineáris modellt hoz létre olyan együtthatókkal, amelyek a legjobban minimalizálják a különbség négyzetek összegét az adathalmazban megtalálható célértékek és a jósolt értékek között. [32][33]

A `fit()` metódus egy lineáris modellt fog létrehozni amely illeszkedik az adathalmazra. Az egyik az `X`, amely egy kétdimenziós tömb és a méretei az `n_samples` ami a példák száma, `n_features`

pedig tulajdonságok száma, ezekben találhatóak az attribútumok, valamint a y tömböt amiben pedig az attribútumokhoz tartozó célértékek találhatóak. [32][33]

A polynomiális regresszió valójában egy speciális esete a lineáris regressziónak ezért továbbra is használhatjuk a LinearRegression osztályt, viszont az adatokat más formában kell átadnunk. A PolynomialFeatures osztály fit_transform() metódusával átalakíthatjuk megfelelő alakba az adathalmazunkat. Fontos, hogy átadjuk hogy hányad fokú legyen. Ezt követően egy mátrixot fog generálni, amiben az attribútumok polynomiális kombinációja található meg kevesebb vagy ahányad fokkal inicializáltunk. [32][33]

Saját nem bináris regressziós fa modell

A regressziós fa felépítése során, binárisan osztjuk fel az adathalmazt. Ez a módszer széleskörben használt mivel működőképesnek és megbízhatónak bizonyult. A bináris felosztásnak köszönhetően viszont feltehetőleg több felosztást kell elvégeznünk ahhoz, hogy felépítsünk egy fát mintha nem bináris felosztásokat hajtanánk végre.

Az ID3 algoritmus nem bináris módon hajtja végre az adathalmaz felosztását, hanem aszerint, ami a legjobban csökkenti az entrópiát.[4] Annak ellenére, hogy nem regressziós feladatok megoldására használható a modell egy kitűnő példa számunkra, hogy milyen a nem bináris felosztás. Ezáltal a számítások mértéke is csökken, amely azt jelenti, hogy rövidebb idő alatt végzi el az algoritmus a rá kiszabott feladatot. Ezenfelül a fa nem lesz annyira mély így a szintek száma lecsökken. Ennek köszönhetően, mivel rekurzívan építjük fel a fát ezért, ha csökkentjük a fa mélységét akkor kevesebb memóriára is lesz szükség és így nem futunk bele olyan problémába, hogy elfogy az elérhető memória egy adott ág felépítése közben.

A nem bináris felosztás így nem csak időt spórol számunkra, hanem energiát is, amely kifejezetten drága manapság. Továbbá a környezetre gyakorolt hatás is csökken, amely egy rendkívül fontos szempont. A növekvő mennyiségű adathalmazok pedig tovább hangsúlyozzák a problémát, hogy hatékonyabban és gyorsabban tanítsunk meg egy modellt.

A paraméterek pedig egy újabb mélységgel bővítik a témakört. Sajnálatos módon nem tudjuk, hogy egy bizonyos paraméter milyen hatással lesz a végeredményre, így kénytelenek vagyunk különböző paramétereket kipróbálni, tesztelni. Ez azt jelenti, hogy a folyamat során egy modellt több alkalommal meg kell tanítani.

Mind ezeken felül maga a feladat megoldása miatt is szükségünk lehet a modell többszöri tanítására. Előfordulhat, hogy arra van szükségünk, hogy adaptív legyen, vagy legalább is a rendszeresen elérhető új vagy kibővített adathalmazból képes legyen tanulni. Hogy a frissített modellt minél hamarabb használhassuk és ezáltal akár a fejlesztés gyorsaságát is növeljük, szükségünk van arra, hogy hatékonyabban és gyorsabban tanítsuk a modellünket, mint korábban.

A regressziós fák, mint azt a fenti kutatások is mutatták, még a mai napig használatban vannak és relevánsak. Ráadásul mint azt láthattuk széleskörű problémakört tudunk vele megoldani. Ezen okokból kifolyólag megalapozott, hogy fejlesszünk olyan módszereket melyek gyorsítják a modell tanítását.

A probléma, amit meg szeretnénk oldani az egy optimalizálási, hibacsökkentési probléma. Annak érdekében, hogy ezt sikeresen megtegyem egy új módszert szeretnék javasolni, mégpedig a nem bináris regressziós döntési fát.

Nem bináris regressziós fa

A hiba mérésére problémától függően kell egy függvényt alkalmaznunk. Amennyiben egy szimpla regressziós fát használunk az alábbi függvényt használhatjuk a hiba mérésére:

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

Amennyiben egy függvényosztály alapú döntési fát használunk, akkor valójában nem az adott $e \in E$ él halmazában található elemeket szeretnénk megvizsgálni, hanem az ahhoz rendelt függvényt. Így a hiba függvény a következő képen írható:

$$H = \sum_{i=1}^n [(ax + b) - y_i]^2$$

Ahol az $(ax + b)$ az $e \in E$ él-hez tartozó lineáris függvény y_i pedig az i -edik adatelem.

Hogy gyorsítani tudjuk a tanítás folyamatát azt többféleképpen is elérhetjük. Egy hatékonyan megírt kód segíthet ezen problémán, erre viszont alapvetően is érdemes törekedni. A legjobb eredményt viszont akkor érhetjük el, ha magát a számítások mennyiségét csökkentjük. Ezt megtehetjük azzal, hogy nem binárisan osztjuk fel adathalmazunkat. Fontos viszont, hogy egy

megfelelő módszert alkalmazzunk. Mint azt az ID3 esetében is láthattuk, hiába volt egy módszer arra, hogy a megfelelő felosztást hajtsuk végre az adathalmazunk alapján, bizonyos esetekben így is előfordult, hogy tévesen hozott döntést. Emiatt pedig szükség volt az algoritmus továbbfejlesztésére a C4.5 keretében [5].

Annak érdekében, hogy egy jól működő modellt tudjunk létrehozni, szükségünk van a megfelelő paraméterekre, amelyek, nem csak a modell pontosságát határozhatják meg hanem lényeges funkciók működését állíthatjuk velük.

Az egyik legfontosabb paraméterünk az az, hogy a bináris felosztás helyett hány részre szeretnénk a felosztásokat elvégezni. Ez akár javíthatja akár csökkentheti a modell pontosságát, valamint azt is könnyedén befolyásolhatja, hogy mennyi idő alatt végez az algoritmus a tanítással.

Továbbá arra is szükségünk van, hogy milyen lépésmérettel végezzük el a felosztást és hogy az hogyan arányúljon az attribútum értékeihez. Nyilvánvalóan, ha az attribútumok 1 és 10 között vesznek fel értéket akkor más lépésméretet kell használni mintha 1 és 100 között vennének fel értékeket.

Az is előfordulhat, hogy hiába értük el a minimális lépésméretet, elképzelhető, hogy van értelme ugyanazzal a lépésmérettel még néhány alkalommal elvégezni egy felosztást. Viszont, ha ez a szám túl nagy akkor felesleges számításokat végezhetünk.

A bináris felosztás során egy fix szám szerint osztjuk fel az adathalmazunkat még az ID3 esetén ez a szám nincs lefixálva. Az, hogy egy adathalmaz esetén amely folytonos célértékekkel rendelkezik nehéz megállapítani hogy melyik mélységi szinten mi lenne a megfelelő számú felosztási szám. Ezért célravezetőbb lehet hogyha ezt a számot egy paraméterként kezeljük.

A felosztási számot jelölje: F

A bináris felosztás azért kézenfekvő mivel egy értéket kell meghatároznunk arra, hogy hol osszuk két részre az adathalmazunkat. A nem bináris felosztás esetén viszont ez egy komplikáltabb és kevésbé egyértelmű feladat lesz, ami miatt extra számításokat kell majd elvégeznünk, viszont összességében még így is szignifikánsan le fog csökkenni a tanítási idő.

A felosztási határok meghatározására egy HAC alapú módszert szeretnék bemutatni, amely számot pedig jelölje: F_h

A módszer lényege, hogy az adathalmazt egy adott attribútuma alapján felosztom részhalmazokra(klaszterekre) majd a HAC -hez hasonlóan összevonom a szomszédos klasztereket, mely végül megadja az adott szinten történő felosztást az F_h -it.

A módszer először megvizsgálja az attribútumokat. Szükségünk van az intervallumra ahhoz, hogy megtudjuk határozni az F_h -t. Ezt követően kiszámítjuk az intervallum felső és alsó határának különbségét a következő képlettel:

$$K = \max_{interval} - \min_{interval}$$

Ahol a $\max_{interval}$ az intervallum felső határa a $\min_{interval}$ pedig az alsó határa. Ezt követően pedig meg szeretnénk határozni azt, hogy mekkora lépésközönként osszuk fel az adathalmazt. Itt fontos odafigyelni, hogy ne legyen túl alacsony ez a lépés szám különben feleslegesen sok számítást végezhetünk. Ezért érdemes meghatározni egy minimum lépés méretet. Erre azért is van még szükség mert ahogy felosztjuk az adathalmazunkat egyre kisebbek lesznek az intervallumok és vele együtt a lépés szám is. Egy bizonyos felosztás után pedig nem vezetne eredményre, ha tovább osztanánk fel az alapján az attribútum alapján (továbbá több üres halmaz jöhet létre). Így a lépés méret egy remek indikátora lesz annak, hogy egy attribútumot mikor ne használjunk fel többet.

Ahhoz, hogy végre tudjuk hajtani a felosztást szükségünk van arra hogy sorrendbe rendezzük az adathalmazunkat így könnyedén képesek leszünk felosztani azt szimplán végig iterálva rajta a lépésméretenként.

A lépés méret meghatározása után pedig végrehajthatjuk az adathalmaz klaszterekbe való felosztását, amely a C halmaz elemeit fogja tartalmazni.

Ezen klasztereket jelölje F_k

Egy klasztert pedig jelöljön F_{k_i}

A lépés méret meghatározására a K értéket használjuk fel.

A lépés méretet jelölése és kiszámítása:

$$L = K/m$$

Ahol L a lépésméret és m pedig egy paraméter, amit az adathalmazunk alapján érdemes megválasztani. Lehetőség szerint tovább javíthatja az algoritmus pontosságát, ha az m paramétert minden attribútumhoz külön választjuk meg.

A minimális lépés méretet jelölje: L_{min}

Például: Az adathalmazunk A_i részhalmaz elemei az alábbiak $a_1, a_2, a_3 \dots a_{n-1}, a_n$. Az A_i elemei növekvő sorrendben helyezkednek el ezt követően pedig meghatározzuk a K értékét. $K = a_n - a_1$. Ezt követően pedig meghatározzuk az L számot. Végül pedig végig iterálva az A_i részhalmazon klaszterekbe osztjuk C halmaz elemeit

Miután mindezt megtettük készen állunk a következő lépésre. Ebben a pontban a pontosság kiértékelésétől függően eltérhet az összevonások meghatározása. Amennyiben a szórást használjuk a pontosság mérésére valamivel egyszerűbb a feladat. Szimplán annyit kell megtennünk, hogy a szomszédos klaszterek közül a két legközelebbit összevonjuk. Mindezt annyiszor megismételjük míg el nem érjük az előre meghatározott F számú klaszter számot.

Amennyiben pedig egy függvény osztályt, azon belül is a lineáris regressziót használjuk a pontosság mérésének alapjául már komplikáltabb a helyzet. Ahhoz, hogy megtudjuk határozni azt, hogy mely klasztereket vonjuk össze előbb a klaszterben lévő adatokhoz hozzá kell rendelni egy lineáris modellt majd annak az hiba függvényéből kiindulva meghatározhatjuk azt hogy mely klasztereket vonjuk össze.

Az hiba függvény, amit használni fogunk az a négyzetes különbség. Először is deriváljuk az hiba függvényt. Ez azért szükséges mert akkor lesz az hiba a legkisebb amikor a függvény deriváltja nulla. Így a derivált függvény fontos számunkra. Az egyváltozós lineáris függvény a következő képen nézz ki.

$$\hat{y} = ax + b$$

Az hiba függvény pedig a következő képen írható fel:

$$H = \sum_{i=1}^n [(ax + b) - y_i]^2$$

Ezt felhasználva a függvényt deriválhatjuk az a valamint b paraméter szerint.

A függvény a szerinti deriváltja:

$$\frac{\partial}{\partial a} = \sum_{i=1}^n (2ax_i^2 + 2x_i b - 2x_i y_i)$$

Ezt levezetve a következő eredményt kapjuk:

$$\frac{\partial}{\partial a} = \sum_{i=1}^n (ax_i^2 + x_i b - x_i y_i)$$

Ezt átrendezve pedig:

$$\frac{\partial}{\partial a} = a \sum_{i=1}^n x_i^2 + b \sum_{i=1}^n x_i - \sum_{i=1}^n x_i y_i$$

A függvény b szerinti deriváltja:

$$\frac{\partial}{\partial b} = \sum_{i=1}^n (2ax_i + 2y_i - 2b)$$

Ezt átrendezve pedig:

$$\frac{\partial}{\partial b} = a \sum_{i=1}^n x_i + \sum_{i=1}^n y_i - \sum_{i=1}^n b$$

Kiemelve a következő részeket:

$$S_1 = \sum_{i=1}^n x_i$$

$$S_2 = \sum_{i=1}^n x_i^2$$

$$S_3 = \sum_{i=1}^n x_i y_i$$

$$S_4 = \sum_{i=1}^n y_i$$

Végül két klaszter közötti „távolságot” megkapjuk az alábbi képlet alapján:

$$D_{i,j} = \left(\frac{S_1}{N} - \frac{S_1}{N}\right)^2 + \left(\frac{S_2}{N} - \frac{S_2}{N}\right)^2 + \left(\frac{S_3}{N} - \frac{S_3}{N}\right)^2 + \left(\frac{S_4}{N} - \frac{S_4}{N}\right)^2$$

Ahol $D_{i,j}$ az i és j -ik klaszter közötti távolság az N pedig az adatok száma.

Azt, hogy egy függvénynek hány minimuma van az viszont egy fontos tényező. Amennyiben az egyenletrendszerünknek egy megoldása van akkor az azt jelenti, hogy egy minimum van az pedig a globális minimum. Egy lineáris egyenletrendszer kompatibilis, ha van megoldása és határozott ha csak egy megoldása van. Ha felírjuk az együttható mátrixot amelyet jelöljön M_e és a M_e^b -t akkor ha

$$\text{rang}(M_e) = \text{rang}(M_e^b)$$

Akkor kompatibilis, és ha

$$\text{rang}(M_e) = \text{rang}(M_e^b) = n$$

Ahol n az ismeretlenek száma akkor kompatibilis és határozott.

$$M_e^b = \begin{pmatrix} a & b & 0 & -1 & 0 \\ 0 & a & 2 & 0 & 0 \end{pmatrix}$$

$$M_e = \begin{pmatrix} a & b & 0 & -1 \\ 0 & a & 2 & 0 \end{pmatrix}$$

Mint láthatjuk

$$\text{rang}(M_e) = \text{rang}(M_e^b) = n = 2$$

Ezt követően, amely két szomszédos klaszter között a legkisebb a $D_{i,j}$ értéke azokat egyesítjük egy klaszterbe. Ezt követően egy új lineáris modellt hozunk létre ennek a klaszternek az adatai alapján. Ezt követően pedig megint végig vizsgáljuk addig a klasztereket míg el nem érjük az F számú klaszter számot. Végül pedig ez lesz a felosztás maga, a következő mélységi szinten.

Mivel csökkentek az adatok mennyisége ezért az intervallumot újra meg kell állapítani és a lépés méretet is. Amennyiben elértük a minimum lépés méretet az adott attribútum alapján már nem

végzünk el több felbontást. Miután minden attribútumot felhasználtunk a maximális alkalommal azt követően véget ér az adott ágon a fa építése.

Inkrementális tanítás

Az inkrementális tanításnak az az előnye, hogy ahelyett, hogy folyamatosan újra kellene tanítani a modellt amint jelentősen megnövekedett az adathalmazunk mérete, inkább részekre bontjuk az adathalmazt és kisebb csomagok segítségével tanítjuk a modellt. Az internetnek és az okoseszközöknek köszönhetően jelentős mennyiségű adat keletkezik napról napra,óráról órára. Így szükségünk van egy olyan módszerre, amely képes magát hozzá igazítani az új adathalmazhoz.[34]

A következő részben egy módszert szeretnék javasolni a döntési fák inkrementális tanítására.

Először is fontos leszögezni, hogy hogyan nézz ki az adathalmazunk. Az inkrementális tanítás során meghatározzuk, hogy a tanításra használt adathalmaz mekkora részét szeretnénk ténylegesen tanításra használni majd azt felosztjuk egy az általunk helyesnek vélt elemszámmra.

Ezen részhalmazokat jelölje: $T_1, T_2, \dots, T_n$

Egy másik megoldás esetén szintén felbontjuk a T tanítóhalmazunkat viszont azzal a különbséggel hogy az egyes részhalmazok elemszáma egy általunk megválasztott mértékben növekedik.

A T_i halmaz számosságát jelölje $dom(T_i)$. A növekedés mértékét pedig I . Ez azt jelenti, hogy a T_1 halmaz $dom(T_1)$ -e az I -vel fog megegyezni. A T_2 halmaz $dom(T_2)$ pedig az $2I$ -lesz.

Az adathalmazunk mérete viszont lehet változó is, vagyis szabály nélküli. Ez azt jelenti hogy az egyik alkalommal például a T_i elemszáma 2000 még a másik alkalommal csak 500. Ebből viszont az következik hogy nem fognak megegyezni az adathalmazok súlya. Így, ha megfelelően nagy az adathalmazunk akkor érdemes lehet a döntési fa egy ágát újra felépíteni, ellenkező esetben viszont ronthatja az eredményeinket. Ha egy nagyobb adathalmaz használatával pontosabb eredményt érhetünk el.

Ezt követően az eddigi, általunk használt módszer szerint megtanítjuk a modellt a T_1 halmazt használva. Most már rendelkezésünkre áll egy kész regressziós fa a probléma vele, hogy egy lényegesen kisebb adathalmazon tanult, mint ami rendelkezésre állt. Éppen ezért a fa különböző szintjeit megvizsgáljuk a következő elérhető adathalmaz segítségével.

A kritérium, ami eldönti, hogy újra építsük-e egy ágat a fában a következő. Az esetek egy részében azonos számú részhalmazokra bontjuk az új és a régi tanító halmazunkat. Ez lehetővé teszi számunkra, hogy összehasonlítsunk két esetet.

Az adott szinthez tartozó $e \in E$ éleket kiértékeljük, hogy mennyire ad pontos eredményt. Ezt úgy tesszük meg hogy vesszük a már létező szinten található felosztást és a hozzájuk tartozó lineáris modelleket. Majd ugyan ezt meg tesszük az újonnan felosztott adathalmazzal. A következő lépésben pedig az F_k klasztereket sorszámaik alapján összehasonlítjuk úgy hogy mind a kettő klaszterre mind a kettő pontosságát megmérjük. Amennyiben a régi felosztás összességében jobban teljesített mint a új akkor nem változtatunk semmin megüünk a következő szintre. Ellenkező esetben viszont újraépítjük a fa azon ágát az új adathalmaz alapján hiszen mind az eddigi mind az új adathalmazra nézve pontosabb eredményt adott összességében. Ezt a folyamatot addig folytatjuk még el nem értünk a levélíg.

Tesztek

Annak érdekében, hogy tesztelni tudjam ezt az új módszert valamint hogy össze tudjam hasonlítani a már meglévő módszerekkel és implementációval különböző adathalmazokat használtam hogy a lehető legpontosabb rálátást kapjam arról hogy mennyire pontosak és gyorsak az eltérő megoldások.

Blob

Az első adathalmaz amit felhasználok, az nem egy előre összerakott emberek által összegyűjtött adatelemekből álló halmaz hanem egy véletlenszerűen generált. Ennek megvalósítására pedig a sckiti-learn, make_blobs metódusát használom. Ennek köszönhetően tudok generálni izotropikus Gaussi cseppet(blob). Ez alapvetően klaszterezésre lett kitalálva, viszont mivel az új módszer, amit használunk részben klaszterezésen alapul ezért elsőre azt várhatnánk hogy ezen az adathalmazon kifejezetten jól fog teljesíteni, a többi módszerhez képest. Az adat generálás során meg tudjuk adni, hogy mennyi attribútummal rendelkezzenek a célértékeke, valamint hogy mennyi középpontot hozzon létre. Mindezen felül még a szórás mértékét is megadhatjuk.

Én elem számnak a 10000-et választottam, központok számának az 5-öt, az attribútumok számának pedig 4-et. A szórást pedig 1-nek választottam.

Ezt az adathalmazt jelölje T_{blob}

Iris

A következő tanító halmaz az iris. Ezt azért választottam mert annak ellenére hogy nem egy nagy vagy bonyolult adathalmaz viszont kiválóan használható többféle módszer és algoritmus által. Továbbá azok, akik elkezdenek mesterséges intelligenciával foglalkozni nagy valószínűséggel bele fognak futni ebbe az adathalmazba, szóval sokak által ismert, továbbá az UCI oldalon legtöbbet letöltött, szóval ebből adódóan van relevanciája ismertsége miatt.

Az adathalmaz 4 attribútumot a virágszirom és a csészelevél hosszát, valamint szélességét tartalmazza centiméterben. Ezen attribútumai valós értékeket vesznek fel. Viszont ez egy osztályozásra használják így egy kis változtatást tettem a célértékekben. A három osztályt, amivel rendelkezik lecseréltem számokra és így már alkalmas regresszióra. 150 tesztből épül fel.

Ezen adathalmazt jelölje: T_{iris}

Wine quality-white

Ezen adathalmaz 12 attribútummal rendelkezik amelyek valós értékeket vesznek fel. Továbbá 4898 tesztből épül fel. A leírás szerint az adathalmaz használható mind osztályozásra mind regresszióra, szóval számunkra megfelelő. Továbbá mivel a többi adathalmazhoz képest lényegesen több attribútummal rendelkezik így láthatjuk hogy képes-e kezelni ezen eseteket. Ezen adathalmazt is az UCI weboldaláról töltöttem le.

House

Ezt a tanító halmazt én állítottam össze és 10000 elemet tartalmaz. Összesen 4 attribútummal rendelkezik és mindegyik attribútumhoz random választottam értékeket a hozzájuk kitalált függvények segítségével. Az első attribútumhoz, ami a méret 0-tól 100-ig választok véletlenszerűen egy értéket egy tizedesjegy pontossáig, majd ezt követően egy függvény segítségével állapítom meg hogy miként befolyásolja a célértéket, amelyet jelöljön P . A függvény a következő:

$$P_1 = \sqrt{\text{méret}} * 10 + \frac{0.3 * \text{méret}}{10}$$

A következő attribútum a „távolság”, amely értéke 30-tól 100-ig tehető és egész értékeket vehet fel. A P -t a következő függvénnyel befolyásolja

$$P_2 = \frac{1000}{\text{távolság}} + 5$$

A harmadik attribútum, amely a „támogatás” amely 0-tól 10-ig vehet fel egész értékeket. A P -t a következő függvénnyel befolyásolja:

$$P_3 = \text{támogatás}$$

A negyedik attribútumom, amely a „keresett” amely 3-tól 8-ig vehet fel egész értékeket. A P -t a következő függvénnyel befolyásolja:

$$P_4 = \text{keresett}^3$$

$$P = P_1 + P_2 + P_3 + P_4$$

Ezen függvények együttesével egy nagy variációval rendelkező adathalmazt kapunk.

Az adathalmazokat a beolvasást követően felosztottam két részre. Minden esetben a 80%-át használtam fel tanításra és 20%-át pedig tesztelésre. Ezzel a módszerrel elkerülhettem azt, hogy azokon az adatokon teszteljek amelyeket már ismer a modell szóval a tesztelés egy valósabb képet fog adni. Ehhez a `sckiti-learn sklearn.model_selection` -ből a `train_test_split` függvényt használtam. Így generáltam négy darab új halmazt mindegy egyes esetben. Két teszt, valamint két tanító halmazt hoz így létre a már korábban említett 80-20 arányban. Mind a két esetben egy attribútum valamint egy célérték halmaz jön létre.

Implementációs környezet

Ahhoz, hogy implementáljam eme új módszert a python programozási nyelvet választottam. Ennek az oka hogy több olyan könyvtár készült a nyelvhez, amit feltudtam használni az implementációm elkészítéséhez, továbbá a nyelv egyszerűbb szintaktikája is segítette a gördülékenyebb implementálást. A már fentebb említett `sckiti-learn` pythonban implementált könyvtárait használtam fel. A lineáris modellek elkészítéséhez szintén a `sckiti-learn`

Linear_regression függvényét használtam. Az idő mérésére a time könyvtárat, valamint a számítások és tömbök kezelésére a math és numpy könyvtárakat vettem igénybe.

A kód megírásához a python 3.7-es verzióját használtam. A jupyter nootbook-ban történt az implementálás és a tesztek lefuttatása is.

Implementációim

Minden implementációra elmondható, hogy először beimportáltam az adathalmazokat elvégeztem a tesz és tanító halmazokra bontást a már fentebb leírt módon. Az algoritmusnak egy-egy attribútum és egy célváltozóból álló halmazt adok át. Mindezek mellett meghatározom a maximális mélységi szintet, valamint azt hogy mi az a halmaz számosság amelyet ha elér azt nyilvánítsa levélnek. Ezek mellett még azt vizsgálom, hogy mi az a hiba érték amelyet ha elér már ne folytassa a felosztásokat halmaz méretétől függetlenül, mivel számunkra az már kielégítő eredményt ad. Végül pedig teszt halmazomat felhasználva megmérem, hogy mennyire pontos még nem láttat adaton valójában a létrejött modellem. Végül a futáskor mért időt, amely a modell tanításának ideje valamint a hiba mértékét eltárolom egy szimpla szöveges fájlban.

Elsőnek egy egyszerű bináris regressziós fát implementáltam. Itt létre hozok egy osztályt, amelyet Node-nak nevetem el. Ebben az osztályban tárolom az összes értéket, amely az adott csomópontra jellemző. Eltárolom, hogy melyik az az érték, amely alapján ketté osztottam az adathalmazt valamint hogy melyik az az érték mely attribútumhoz tartozik. Az, hogy mely ághoz tartozik a Node, azt a sorrend határozza meg. A következő két Node objektumot, amelyet képzünk egy tömbben tárolom el úgy hogy sorrendben legyenek és így nincs szükség segédváltozókra.

Ahhoz hogy meghatározzam hogy melyik az az attribútum amely alapján érdemes a felosztást elvégezni és hogy melyik az az érték ahhoz a find_best_split() függvényemet használom. Ebben a már fentebb leírt technikával végig iterálok az attribútumhoz tartozó értékeken és a calculate_split_std függvény segítségével megállapítom, hogy mennyi lesz a hiba mértéke. Ahhoz hogy kitudjam választani az adott attribútumhoz tartozó értékeket az általam írt get_features() metódust használom. Majd miután megtaláltam a legkissebet azt kiválasztva elvégzem a felosztást, és rekurzívan addig folytatom még a már megfogalmazott kritériumokat teljesíti az adathalmaz.

Azok a döntési fákat, amelyek függvényosztály alapúak ott annyit változtattam a programon hogy a hiba függvény már a lineáris modellekre értendőek.

A HAC alapú felosztás során pedig azt is meghatározom, hogy mi legyen a felosztás szám

Paraméterek

Mindegyik implementációnál meghatározok egy értéket, amely megmondja, hogy mennyi az a hiba mérték amelyet követően már nem végzünk el több felbontást mivel számunkra már megfelelő pontosságot add vissza. Továbbá meghatározom azt a számot is, amely egy fa levelének legkiseb elemszámát adja meg. Ez azt jelenti, hogy ha egy csomópont elemszáma kisebb ennél az értéknél akkor szintén nem folytatódik az az ág tovább és azt a csomópontot levélnek nyilvánítjuk. Továbbá meghatároztam egy maximum mélységet, amelyet túllépve már nem épül tovább a döntési fa és az azon a mélységen található csomópontokat levélnek nyilvánítjuk. Ilyen határt viszont csak abban az esetben határoztam meg amikor szükséges volt mivel más különben a rekurzió miatt a nem megfelelő memória mennyiség okán nem lett volna képes a számítógép befejezni a modellt.

Mérések szervezés, mérőszámok

Arra, hogy megmérjem a modellek pontosságát egy a MSE függvényt használok. Ez a következő képen írható fel:

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

Ahol N az elemszám, $\hat{y}_i$ a modell által jóslott érték, y_i pedig a valós érték. Minden implementáció esetén

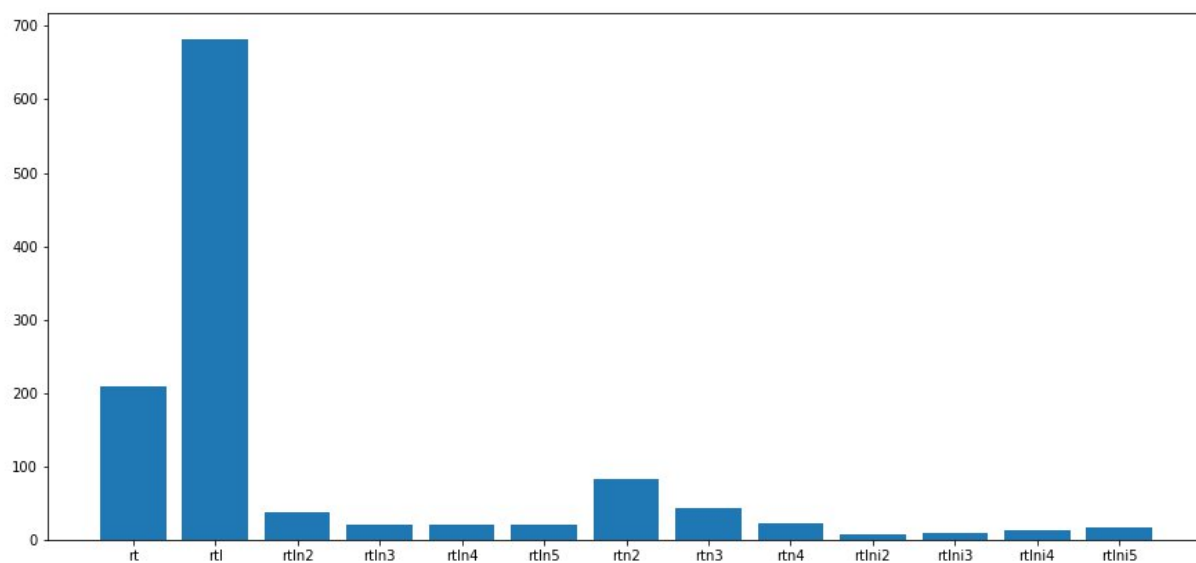
Minden algoritmus 5 alkalommal futtatam le ugyan azon az adathalmazon majd vettem a hiba és az idők átlaga. Továbbá a nem bináris megoldást használó algoritmusok esetén több felosztás számot is megvizsgáltam. Ugyan, amely módszert bemutatam annak az volt a lényege hogy nem binárisan osztom fel az adathalmazt úgy gondoltam érdekes eredményekhez vezethet ha az új módszer segítségével binárisan is elvégzema felosztásokat.

Az időt pedig másodpercekben mértem 15 tizedesjegy pontossággal.

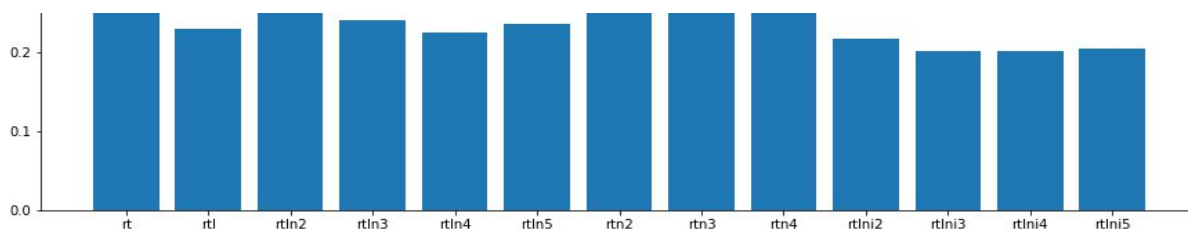
A különböző megoldások jelölései

Módszer	Felosztás szám	Jelölés
---------	----------------	---------

Regresszió fa	2	rt
Függvényosztály alapú Regressziós fa	2	rtl
Függvényosztály alapú nem bináris Regressziós fa	2	rtln2
Függvényosztály alapú nem bináris Regressziós fa	3	rtln3
Függvényosztály alapú nem bináris Regressziós fa	4	rtln4
Függvényosztály alapú nem bináris Regressziós fa	5	rtln4
Nem bináris Regresszió fa	2	rtn2
Nem bináris Regresszió fa	3	rtn3
Nem bináris Regresszió fa	4	rtn4



1.ábra. Blob adathalmaz tanítási ideje másodpercben mérve



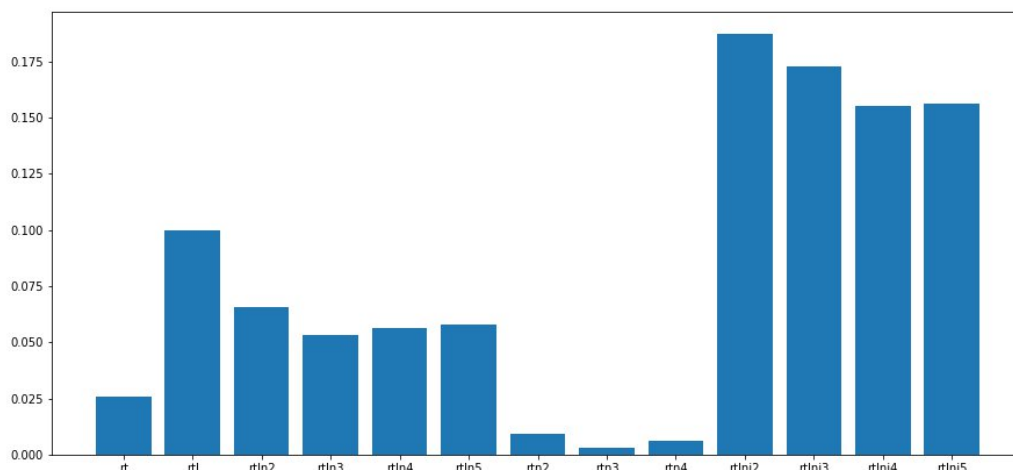
2.ábra. Blob adathalmaz hiba mértéke

Blob eredményei

Mint azt az első ábra is mutatja a különböző módszerek rendkívül eltérnek időben. Az *rt* lényegesen gyorsabban végezte el a tanulást, mint az *rtl*. Ez nem is annyira meglepő mivel az *rtl*-nek még lineáris modellek is létre kell hoznia így ez megnöveli mind a szükséges számítások számát mind az időt, ami alatt végez. Ami viszont egyből feltűnik az olvasónak, hogy a nem bináris felosztásokat használó módszerek rendkívül gyorsak még abban az esetben is, amikor binárisan végzik el a felosztás. Ami még egy fontos észrevétel, hogy azáltal, hogy megnöveltük a felosztások számát csökkent a tanítási idő is. Ez az *rtn* esetében sokkal jobban kijött az *rtl*-el szemben. Az *rtlni* is rendkívül gyorsan végzett. Viszont hiába gyors a modell tanítása, ha nem tud pontos eredményt adni. Így fontos, hogy azt is megvizsgáljuk.

Az második ábra ami mutatja a hiba mértékét. Ezen azt láthatjuk, hogy az *rtl* amelynek a legtovább tartott a tanítása teljesített a legjobban. Viszont nem annyival jobb, mint amennyivel lassabb. Az *rtln* esetén láthatjuk, hogy nem csak hogy rendkívül gyors de még pontos is. Ugyan nem olyan pontos, mint az *rtl*, de egy nagyobb adathalmaz esetén már lehet ez a mutató is a nem bináris felosztás javára lenne. Mint láthatjuk egy kevés pontosságot veszítünk cserébe nagyságrendekkel gyorsabban végez a döntési fa kiépítésével az algoritmus.

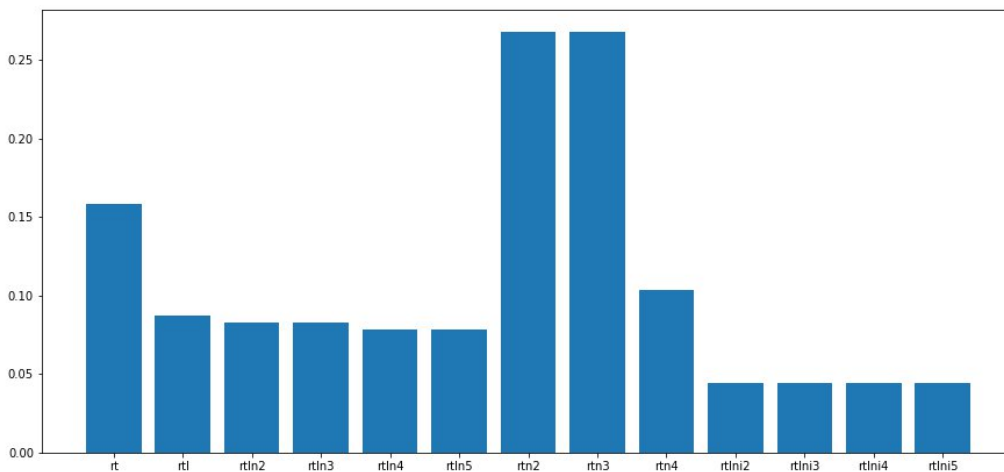
A blob adathalmaz esetén az új módszer jól vizsgázott. Teljesítette az előre kitűzött célt hogy nem csak gyorsabb legyen de ezért ne veszítsünk a pontosságból.



3.ábra. Iris adathalmaz tanítási ideje másodpercben mérve

Iris eredményei

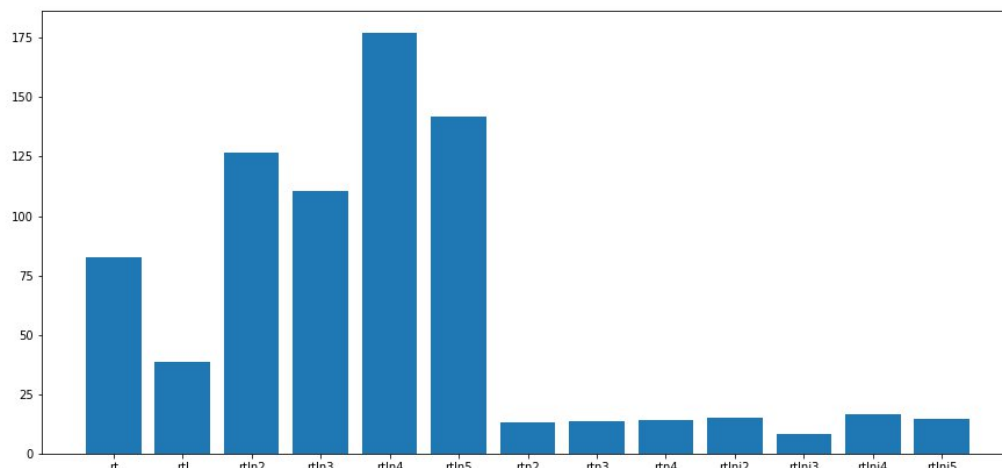
Az iris adathalmaz esetén fontos megjegyezni, hogy ez egy rendkívül kicsi adathalmaz a többihez képest. Ez nem probléma, hiszen, ha a módszer kevés adatból is viszonylag pontos eredményt tud adni akkor ez azt jelenti hogy nem csak az adathalmaztól, de mérettől függetlenül is jól tud teljesíteni. A kevés elemszámból adódóan abban az esetben amikor függvény osztályt is használunk az megnöveli a számítások számát. A Blob adathalmaz esetén mégis a nem bináris



felosztás esetén tudott kifejezetten gyors lenni. Ez az effektus azért nem tud teljesülni mivel minél több felosztás szükséges a bináris módszernek, annál jobban fog teljesíteni időben a nem bináris módszer és a pontosság miatt megéri nagy adathalmazokra nézve bevállalni ezt az extra számítást.

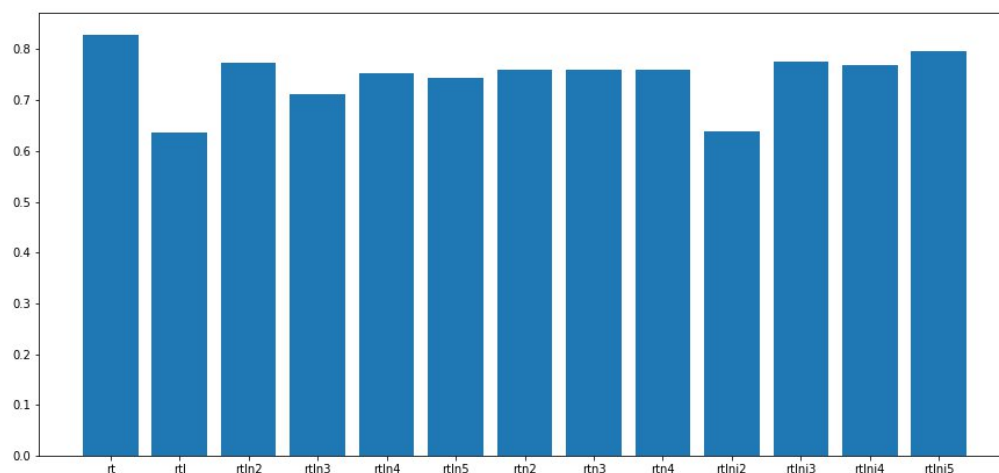
Pontosság szempontjából láthatjuk, hogy a szimpla nem bináris felosztás hiába volt a leggyorsabb rendkívül pontatlan. A lineáris modellt használó nem bináris felosztás ugyan lassabb volt cserébe viszont pontosabb is, feltételezésem szerint mivel a kevés adatból adódott zajt valamennyire ellensúlyozta a lineáris modell használata.

Wine eredményei



5.ábra. Wine adathalmaz tanítási ideje másodpercben mérve

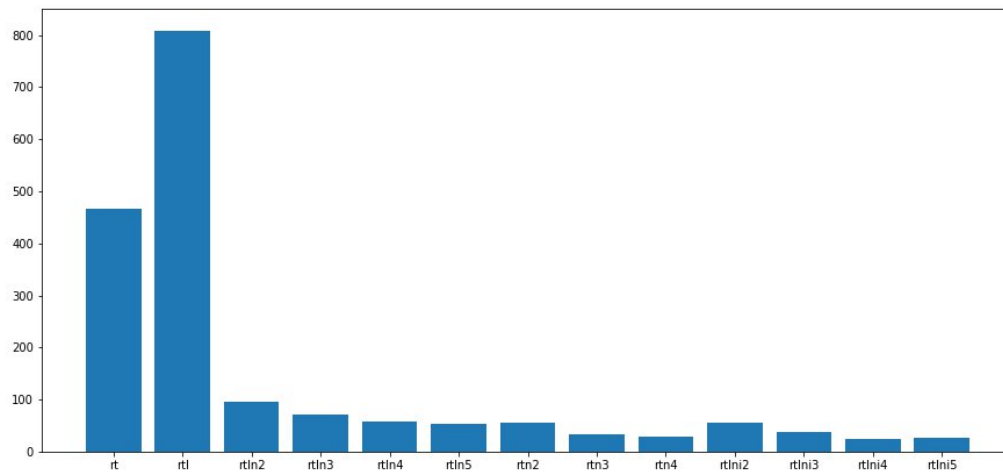
A wine dataset rendkívül érdekesen viselkedik. Számítási időben, a nem bináris felosztás megint csak jeleskedett, amely tovább erősíti eme módszer hasznosságát. Itt láthatjuk, hogy 13 attribútum esetén is rendkívül gyors viszont a lineáris modell használata ebben az esetben nem tűnik kifizetődőnek, valószínűleg ugyanabból az okból. A legtöbb számítási időt veszi igénybe. Fontos viszont, hogy milyen pontosságot értek el. Hiába gyors az algoritmus, ha pontatlan.



6.ábra. Wine adathalmaz hiba mértéke

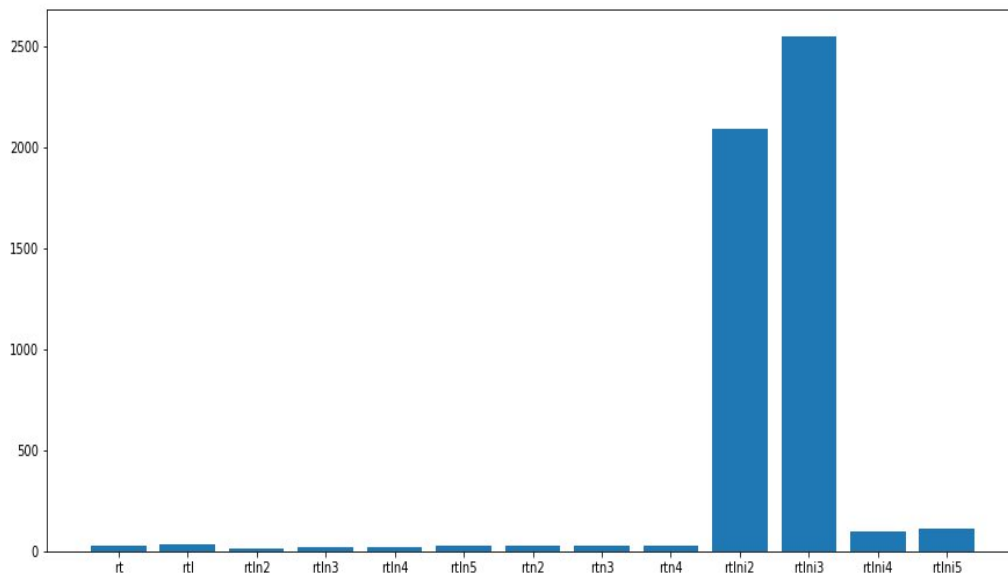
Meglepő módon nincsenek kimagasló különbségek a pontosságban. Így az idő, ami alatt kiépítjük a döntési fát, sokkal nagyobb súlyú. Ebből kifolyólag futási idő pontosság arányában a nem bináris felosztás módszere jeleskedett. Hasonlóan jól teljesített a szimpla lineáris modellt használó algoritmus viszont, nem teljesít olyan konzisztensen jól, mint a nem bináris felosztás.

House eredményei



7.ábra. House adathalmaz tanítási ideje másodpercben mérve

A futási idő megint sokat mondó. Az eddig használt sima bináris felosztás borzasztóan rosszul teljesített a nem bináris felosztás módszeréhez képest. A lineáris modell használata megint növelte a futási időt a kérdés már csak annyit, hogy cserébe mennyi pontosságot nyerünk. Eddig bármely kategóriát is néztük a nem bináris felosztás módszerének valamely variációja teljesített a legjobban.



8.ábra. House adathalmaz hiba mértéke

Ahogy eddig tapasztaltuk és várható volt a lineáris modell használata ténylegesen növelte a pontosságot. Ebben az esetben a felhasználó feladata eldönteni, hogy számára megéri-e az extra pontosság plusz tanítási időért cserébe. Az rtlni viszont kiemelkedően rosszul teljesített két esetben különösen.

A nem bináris felosztás minden adathalmazon jól teljesített. Vagy időben vagy pontosságban vagy akár mind a kettőben kiemelkedő teljesítménnyel rendelkezett. Adathalmaz méretétől és komplexitásától függően láthatjuk az eredményeket, amelyek azt bizonyítják, hogy a nem bináris felosztás módszere igen is hasznos és jól is tud teljesíteni.

Összefoglaló

A döntési fa a mesterséges intelligencia területén hosszú történelemmel rendelkezik, és így egy régóta, de még mai is használt módszer. Mint láthatuk annak ellenére, hogy a legelső döntési fáról szóló cikket már 1963-ban publikálták [7], még egy mai is használt módszer, mint azt a fenti kutatások is mutatják, így van létjogosultságuk. A döntési fákat használhatjuk osztályozási probléma megoldására, amelyre az ID3 algoritmus add egy megoldást. Ezt tovább fejlesztették és így a C45[4] már nem rendelkezik az ID3 minden hibájával. Amennyiben regressziós feladatot

szeretnénk megoldani arra használható több algoritmus is köztük a CART, amely binárisan felosztásokat végez. Az adatelemzésben számtalan módszer létezik, mint például a lineáris regresszió valamint HAC. Ezen módszerek segítségével amelyeknek több implementációja is létezik, például a scikit-learn-től, segíthetik más algoritmusok továbbfejlesztését optimalizálását. A függvényosztály alapú döntési fa például egy lineáris modell használatával szeretne pontosabb eredményt elérni mely sikerül is [27]. Az általam bemutatott módszer is egy hasonló továbbfejlesztés.

A nem bináris regressziós döntési fa módszerének célja, hogy tovább optimalizálja a már létező módszereket. Egy HAC alapú módszer segítségével végzem el a felosztásokat, amelynek köszönhetőn gyorsabban végezhetünk a fa kiépítésével. Mint azt a tesztek is mutatják a nem bináris felosztásnak köszönhetően jelentősen lecsökkent a tanítási idő úgy, hogy az nem ment a pontosság rovására. Természetesen egy módszer nem képes minden adathalmazra nézve a legjobban teljesíteni. Viszont az egyértelműen körvonalazódik a tesztek eredményeiből, hogy a nem bináris felosztást használó algoritmusok valamelyike teljesített mindig a legjobban. Így elmondható, hogy a nem bináris fa módszere sikeresen optimalizálta a már meglévő megoldásokat.

Jelölések

T - tanító halmaz

En – entrópia

I_g – Information gain

S_i – Split info

G_r – Gain ratio

Z_i – Határpontok

MSE – Mean Squared Error

H – hiba

F - Felosztási szám

F_h - Felosztási határok

K – Intervallum felső illetve alsó határainak különbsége

F_k – Lépésméret alapján kialakított klaszterek halmaza

F_{k_i} - F_k egy klasztere

L – Lépésméret

L_{min} – Minimális lépésméret

Irodalomjegyzék

- [1] Kok, Joost N., et al. "Artificial intelligence: definition, trends, techniques, and cases." *Artificial intelligence* 1 (2009): 270-299.
- [2] Choubin, Bahram, et al. "An ensemble prediction of flood susceptibility using multivariate discriminant analysis, classification and regression trees, and support vector machines." *Science of the Total Environment* 651 (2019): 2087-2096.
- [3] AKINCI, T. Çetin, and H. Selçuk NOĞAY. "Application of decision tree methods for wind speed estimation." *European Journal of Technique (EJT)* 9.1 (2019): 74-83.
- [4] Quinlan, J. Ross. "Induction of decision trees." *Machine learning* 1 (1986): 81-106.
- [5] Andreas C. Muller and Sarah Guido (2016) *Introduction to Machine Learning with Python* O'Reilly Media,
- [6] Morgan, James N., and John A. Sonquist. "Problems in the analysis of survey data, and a proposal." *Journal of the American statistical association* 58.302 (1963): 415-434.
- [7] Loh, Wei-Yin. "Fifty years of classification and regression trees." *International Statistical Review* 82.3 (2014): 329-348.
- [8] Breiman, Leo, et al. "Cart." *Classification and regression trees* (1984).
- [9] Quinlan, J. Ross. *C4. 5: programs for machine learning*. Elsevier, 2014.
- [10] Romeo, Luca, et al. "Machine learning-based design support system for the prediction of heterogeneous machine parameters in industry 4.0." *Expert Systems with Applications* 140 (2020): 112869.
- [11] Zhang, Chen, et al. "Road condition recognition in self-driving cars based on classification and regression tree." *ICIC Express Letters, Part B: Applications* 10.12 (2019): 1115-1122.
- [12] Ktenioudaki, Anastasia, et al. "Blueberry supply chain: Critical steps impacting fruit quality and application of a boosted regression tree model to predict weight loss." *Postharvest Biology and Technology* 179 (2021): 111590.
- [13] http://project.mit.bme.hu/mi_almanach/books/aima/ch18s03

- [14] Peng, Wei, Juhua Chen, and Haiping Zhou. "An implementation of ID3-decision tree learning algorithm." From web. arch. usyd. edu. au/wpeng/DecisionTree2. pdf Retrieved date: May 13 (2009).
- [15]<https://medium.com/analytics-vidhya/decision-trees-for-classification-id3-machine-learning-6844f026bf1a>
- [16]<https://levelup.gitconnected.com/c4-5-decision-tree-explained-from-bottom-up-67468c1619a7>
- [17] Hssina, Badr, et al. "A comparative study of decision tree ID3 and C4. 5." International Journal of Advanced Computer Science and Applications 4.2 (2014): 13-19.
- [18]<https://medium.com/deeplearning-concepts-and-implementation/independent-and-dependent-variables-in-machine-learning-210b82f891db>
- [19] Sebastian Raschka (2015) Python Machine Learning Packt Publishing
- [20]<https://towardsdatascience.com/cart-classification-and-regression-trees-for-clean-but-powerful-models-cc89e60b7a85>
- [21]<https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047>
- [22]<https://www.ibm.com/topics/linear-regression>
- [23]<https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/>
- [24]<https://towardsdatascience.com/polynomial-regression-bbe8b9d97491>
- [25]<https://www.analyticsvidhya.com/blog/2021/10/understanding-polynomial-regression-model/>
- [26]<https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21>
- [27] Quinlan, John R. "Learning with continuous classes." 5th Australian joint conference on artificial intelligence. Vol. 92. 1992.
- [28]<https://towardsdatascience.com/machine-learning-algorithms-part-12-hierarchical-agglomerative-clustering-example-in-python-1e18e0075019>

- [29]<https://scikit-learn.org/stable/modules/tree.html>
- [30]<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
- [31]<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html>
- [32]https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- [33]<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html>
- [34]<https://medium.com/analytics-vidhya/incremental-online-learning-9868861db880>
- [35]French, Robert M. "The Turing Test: the first 50 years." Trends in cognitive sciences 4.3 (2000): 115-122.

Tartalomjegyzék

Bevezetés	2
Felügyelt és felügyelet nélküli tanítás	3
Döntési fák	4
ID3	5
C4.5	7
Regresszió	9
Regressziós fa	9
Lineáris regresszió	10
Polynomiális regresszió	10
Függvényosztály alapú regression tree	12
Hierarchical Agglomerative Clustering(HAC)	12
Meglévő implementációk	12
Döntési fa	12
HAC	13
Linear regression/polynomial regression	13
Saját nem bináris regressziós fa modell	14
Nem bináris regressziós fa	15
Inkrementális tanítás	21
Tesztek	22
Blob	22
Iris	23
Wine quality-white	23
House	23
Implementációs környezet	24
Implementációim	25
Paraméterek	26
Mérések szervezés, mérőszámok	26
Blob eredményei	28
Iris eredményei	29
Wine eredményei	31
House eredményei	32
Összefoglaló	33

Jelölések.....	35
Irodalomjegyzék.....	36

Eredetiség nyilatkozat

"Alulírott Bán Péter, a Miskolci Egyetem Gépészmérnöki- és Informatikai Karának hallgatója büntetőjogi és fegyelmi felelősségem tudatában kijelentem és aláírásommal igazolom, hogy ezt a dolgozatot saját magam készítettem, a benne leírt vizsgálatokat – ha ezt külön nem jelzem – magam végeztem el, és az ismertett eredményeket magam értem el. Adatokat, információkat csak az irodalomjegyzékben felsorolt forrásokból használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

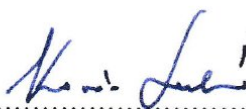
Miskolc, 2023.05.04


.....
a hallgató aláírása

A konzulens nyilatkozata

Alulírott, dr. Kovács Lóránd a Miskolci Egyetem Informatika Intézetének oktatója a Nem bindás megesszés döntés fel című TDK dolgozatot beadásra alkalmasnak ítélem.

Miskolc, 2023.05.04


.....
a konzulens aláírása