Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Távközlési és Médiainformatikai Tanszék

Hidasi Balázs

(HIDASIB@FREEMAIL.HU)

MODELL ALAPÚ IDŐSOR-OSZTÁLYOZÓ FEJLESZTÉSE ÉS KITERJESZTÉSE

Konzulens:

Gáspár-Papanek Csaba

tanársegéd, Távközlési és Médiainformatikai Tanszék

(GASPAR@TMIT.BME.HU)

Budapest, 2011. május

TARTALOMJEGYZÉK

HALLGATÓI NYILATKOZAT

Alulírott **Hidasi Balázs**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2011. 05. 17.

.....

Hidasi Balázs

KIVONAT

Az idősor-osztályozás problémájának megoldására az esetek többségében példány alapú algoritmusokat használnak, amelyek sok előnnyel, de lagalább ugyanennyi hátránnyal rendelkeznek a modell alapú módszerekkel szemben. Bizonyos problémákra léteznek modell alapú megoldások, ám ezek alkalmazása vagy előzetes szakértői tudást követel, vagy csakis az adott problémára lehetséges.

A ShiftTree algoritmus egy olyan, saját fejlesztésű, modell alapú idősor osztályozó, ami szakértői tudás nélkül is az alapvető példány alapú módszerek pontosságával alkalmazható, miközben számos előnyös tulajdonsággal rendelkezik, mint amilyen a modellek értelmezhetősége, vagy a szakártői tudás modellbe építhetősége, vagy a problémafüggetlen alkalmazhatóság. A dolgozatban bemutatom a ShiftTree idősor-osztályozó algoritmust, és megvizsgálom, hogy milyen módon lehetne azt továbbfejleszteni, hogy még hatékonyabbá válljon.

Az algoritmus pontosságát növelendő, kibővítem az algoritmus operátorkészletét, és egy teljesen új operátorcsaládot vezetek be. Megvizsgálom, hogy hogyan lehet úgy megváltoztatni a tanítási algoritmust, hogy a modellek pontossága elérje, a többszörös modellezéssel elérhető pontosságot, de ezért ne kelljen sokszorosára megnőtt futási idővel, illetve modell-komplexitással fizetnünk. A módosításokat 23, különféle területről vett, eltérő tulajdonságú idősor-osztályozási feladaton tesztelem, és az eredményeket a legelterjedtebb szomszéd módszerek eredményeivel is összehasonlítom. Az optimálisnak tűnő módszert összehasonlítom a 2007-es Time Series Challange verseny eredményivel, vak tesztek keretében.

Azonosítom azokat a pontokat, amelyeken az algoritmust optimalizálva a tanítási idő jelentősen lecsökkenthető. Az optimalizálást elvégezve megvizsgálom, hogy ez hogyan hatott a futási időre és az algoritmus skálázódására.

Megvizsgálom, hogy az algoritmust felhasználva milyen összetett modellek készíthetőek. Kétféle kombinálási módszert megvizsgálok: az elterjedt boostingot és a saját fejlesztésű, keresztvalidáción alapuló metódust. Ezeket az úgy nevezett forest eljárásokat 22 adatsoron tesztelem, és az optimálisnak tűnő módszert összehasonlítom a 2007-es Time Series Challange verseny eredményivel, vak tesztek keretében.

Módosítom az algoritmust úgy, hogy képes legyen az osztályozási konfidenciák kezelésére, illetve képes legyen kezelni azt, anélkül, hogy újra kéne modelleznünk, ha az osztályozandó adatok tulajdonságai az idővel megváltoznak. Ez utóbbi a gyakorlati problémák esetében általános jelenség.

Röviden megvizsgálom, hogy az algoritmus mögötti elv, illetve az algoritmus modelljei milyen más - az idősor-osztályozástól eltérő - területen, milyen feltételek mellett alkalmazhatóak, és felvázolom a jövőbeli, lehetséges kutatási/fejlesztési irányokat.

Mindezen fejlesztésekkel elérem, hogy a ShiftTree egy nagyon jó tulajdonságú idősor-osztályozó rendszerré válljon, ami könnyen kiterjeszthető más félig-strukturált, illetve strukturált adatok osztályozására.

Kulcsszavak: ShiftTree, adatbányászat, idősor-osztályozás, döntési fa

ABSTRACT

Most of the commonly used time-series classification algorithms are instance based algorithms. These methods have some advantages but also have at least that many disadvanteges compared to model based methods. Model based time-series classifiers on the other hand are only usable efficiently if we have some prior knowledge on the problem we want to solve or are only efficient in one field of application (e.g. speech recognition).

Teh ShiftTree algorithm is a model based time-series classifier that was developed by me. It has the accuracy of the instance based methods and has every advantage that a model based method can offer and even more like (a) interpretability, (b) generality, (c) prior knowledge can be built into the model. In this thesis I introduce the ShiftTree algorithm and examine how the algorithm can be improved and be made more efficient.

I expand the operator set of the algorithm and introduce a new operator family in order to make the algorithm more accurate. I examine how the training process can be altered in a way that the accuracy may exceed the accuracy of the multiple modeling mode while the running time and the model complexity of the algorithm stays low. I evaluate the modifications on 23 different datasets and I compare them to the results of the most common instance-based algorithms. The optimal method is also compared to the results of the KDD Time Series Challange 2007 using blind tests.

I identify the bottlenecks in the training process in order to optimize the running time of the algorithm. I examine the effect of the optimization on the running time and on the scalability of the algorithm.

I also describe the experiments I have done with assambled models. I introduce two types of the model assambling: one based on the common boosting algorithm and one based on cross validation. I evaluate these assembling or forest methods on 22 different data sets. The optimal method is also compared to the results of the KDD Time Series Challange 2007 using blind tests.

I modify the algorithm in order to make it capable of using labeling confidences and I introduce a method that makes the algorithm able to adapt to the property changes of the input data without having to rebuild the whole model (on-line learning). This is certainly an important feature when using the algorithm on real-life data.

I briefly examine how the principles of the algorithm can be expanded on other fields of semi-structured data mining. I also shortly describe how the ShiftTree models can be used in stream mining.

These improvements make the ShiftTree a very efficient algorithm family in the fileds of semi-structured data mining and esspecially in the filed of time-series classification.

Keywords: ShiftTree, data mining, time-series classification, decision tree

Bevezetés

Az idősorok adatbányászata az utóbbi néhány évben az adatbányászat egyik felkapott kutatási területévé vált. A nevesebb adatbányászati konferenciákon sokszor külön szekció foglalkozik az idősorokkal, a félig-strukturált adatok adatbányászatával, ahová az idősorokkal kapcsolatos feladatok is tartoznak. Ennek oka az, hogy az elmúlt évtizedekben olcsóbbá váltak a különféle szenzorok, így rengeteg idősor-jellegű adat keletkezett, viszont ezeknek az adatoknak az elemzési módszerei eltérnek a klasszikus adatrekordok elemzésétől, mivel az utóbbi módszerek csak korlátozottan használhatóak az eltérő struktúra miatt.

Az idősorokkal elképzelhető feladatok sokfélék, az egyik ezek közül az idősorok kategóriákba sorolása, vagy más néven osztályozása. A dolgozatban erre a problémára mutatok be egy olyan megoldást, ami sok mindenben eltér a területen megszokott megoldásoktól, mégis azokhoz hasonló teljesítménnyel működik, sok szempontból pedig előnyösebb tulajdonságokkal rendelkezik. Ennek a módszernek az alapjait 2007-ben fektettem le a konzulensem segítségével, 2008-ra pedig ennek alapján egy jól működő idősor-osztályozót hoztam létre. Az algoritmus viszont közel sem volt teljes. Ennek a dolgozatnak a fő témája az, hogy milyen fejlesztéseket hajtottam végre az algoritmuson, ami által az sokkal hatékonyabbá vált több szempontból is. A dolgozat másik fő vonala azon kiterjesztéseknek a vizsgálata, amik nem szorosan az alap algoritmushoz kötődnek, de a segítségükkel a módszer javítható és/vagy új képességekre tesz szert. A dolgozat az alábbiak szerint épül fel.

Az 1. fejezetben bemutatom az idősor-osztályozás problémáját, röviden bemutatom a területen elterjedt szomszéd alapú módszereket, felvázolom, hogy miért lehet szükség egy újfajta, modell alapú megközelítésre, és bevezetem a dolgozatban használt jelölésrendszert. A 2. fejezetben ismertetem a munka kiindulási állapotaként felhasznált ShiftTree idősor-osztályozó algoritmust, annak néhány tulajdonságát és eltérő variánsait. A 3. fejezet az első nagyobb fejlesztési lépés eredményét, az algoritmus új operátorait és azok működését írja le. A 4. fejezet két fontos fejlesztési iránnyal foglalkozik. Az egyik a módszer tanító algoritmusának futási idejének a csökkentésére kidolgozott módszerek. A másik a módszer pontosságának növelésére kidolgozott alternatív tanítóalgoritmusok. Ez a fejezet szintén tartalmaz egy részletes szekciót, amiben a különféle variánsok eredményei kerülnek összehasonlításra egymással, és egy elterjedt idősor-osztályozó módszerrel. Az 5. fejezet a legfontosabb kiterjesztést, a több modell kombinálását leíró módszereket és azok tesztjeit foglalja magában. A 6. fejezet egy másik fontos kiterjesztésről, az osztályozási konfidenciáról és az így bevezethető on-line tanulásról ír. A 7. fejezetben egy rövid példán keresztül demonstrálom a módszernek azt az előnyös tulajdonságát, hogy bár szakértői tudás nélkül is kellően pontos, a szakértői tudás könnyen beépíthető a modellbe, miáltal az még pontosabbá válik. A 8. fejezetben egy rövid kitekintés keretében megmutatom, hogy milyen elvek mentén és milyen (az idősor-osztályozástól eltérő) problémák megoldására lehet kiterjeszteni az algoritmust, illetve a mögötte lévő elvet. A dolgozatot egy összefoglaló zárja a 9. fejezetben.

1. Idősor-osztályozás

Ebben a bevezető fejezetben ismertetem az idősor szerű adatokat, és definiálom, hogy a továbbiakban mit értek idősoron. Ismertetem az osztályozás és az idősor osztályozás problémáját, valamint néhány klasszikus, elterjedt megközelítés a probléma megoldására. Szintén ebben a fejezetben adom meg azt a jelölésrendszert, amit a továbbiakban használni fogok.

1.1. Idősorok és osztályozásuk

Az idősorok félig strukturált adatok, amelyek általánosan úgy írhatóak le, hogy egy vagy több időtengellyel és egy vagy több megfigyelt változóval rendelkeznek. Az időtengelyek jelentősége, hogy ha a T_i időtengelyen a $t_{i,1} \leq t_{i,2}$, akkor a $t_{i,1}$ időpillanatban történt megfigyelésről tudjuk, hogy valamilyen módon (pl. térben, időben, stb.) megelőzte a $t_{i,2}$ időpillanatban történt megfigyelést.

1.1.1. Idősorok csoportosítása

Azokat az idősorokat, amik több időtengellyel rendelkeznek, többdimenziós idősornak nevezzük. Egy két dimenziós idősor lehet például egy szürkeárnyalatos fénykép, aminek a két időtengelye az X és Y koordinátatengelyek, a megfigyelt változó pedig az adott pixel világosságának mértéke. Egy három dimenziós példa lehet egy videó, ami az X és Y tengelyek mellett rendelkezik egy T időtengellyel is, megfigyelt változóból pedig hárommal rendelkezik: a vörös, a zöld és a kék színek erőssége az adott pixelben. Az irodalomban változó, hogy a többdimenziós idősorokat is idősoroknak tekintik-e vagy csak a klasszikus, egy időtengellyel rendelkező struktúrákat nevezik idősornak. Ez utóbbira példa lehet a hőmérséklet értéke az idő függvényében.

Az idősorokat csoportosíthatjuk az alapján is, hogy mennyi megfigyelt változójuk van. A több változóval rendelkező idősorokat többváltozós idősoroknak szokás nevezni. Sajnos az angol megnevezés a szakirodalom a multi-attribute mellett sokszor a multi-dimension, így az elején meg kell vizsgálni, hogy a multi-dimension alatt a szerző több dimenziós vagy több változós idősorokat ért. Többváltozós idősor lehet például egy nap a Budapesti Értéktőzsde összes részvényének árfolyama, egyváltozós pedig a BUX index.

Egy harmadik csoportosítási mód az időtengelyen az időértékek alapján lehetséges. Léteznek folytonos idősorok, ahol az időtengely(ek) minden időpillanatában megfigyelhető az idősor változója/változói. Elméletben ilyen a fenti hőmérsékletes példa. Ez a kategória a gyakorlati adatbányászati szempontból nem lényeges, mivel nem tudunk continuum számú megfigyelés eltárolni[1]. Egy másik kategória a tranzakciós idősorok kategóriája. Ebben az esetben néhány időponthoz tartoznak megfigyelések. Ezen időpontok gyakoriságára, eloszlására semmilyen megkötés nincs, mint ahogy arra sem, hogy minden időpontban minden változó megfigyelhető-e. A fenti részvényárfolyamos példa tipikusan ebbe a kategóriába tartozik, amikor a változók az egyes adás-vételek időpontjaiban megfigyelhetőek/értelmezettek. A klasszikus idősorok kategóriájába olyan idősorok tartoznak, ahol az időtengelyen egyenletes időközönként helyezkednek el azok az időpontok, amelyekben megfigyeljük a változók értékeit (több tengely esetén nem szükséges mindenhol ugyanazt a mintavételi frekvenciát alkalmazni). Az egyenletes mintavételezés miatt a pontos időértékek el is hagyhatóak, és az egyes elemeket azonosíthatjuk a tengelyeken elfoglalt helyükkel: pl. az első tengely szerinti

[1] Az ilyen típusú idősorokat le lehet írni (vagy közelíteni lehet) szakaszonként függvényekkel, de ezeket már nem szokás idősoroknak tekinteni. Abban az esetben, ha csak bizonyos időpontokhoz tartozó értékeket tárolunk, akkor az idősor már nem tekinthető folytonosnak.

ötödik és a második tengely szerinti második pozícióban lévő elem. A fenti fényképes és videós példa is ebbe a kategóriába tartozik.

A fentiek mellett elképzelhetőek még egyéb csoportosítások (pl. változók értékkészlete alapján: kategória, sorrendezhető, diszkrét, folytonos, stb.), de számunkra ezek nem fontosak. A továbbiakban idősor alatt az egy dimenziós (egy időtengelyes), egy- vagy többváltozós, egyenletesen mintavételezett, azaz a klasszikus egy- vagy többváltozós idősorokat értem. Mindig utalok rá, ha többváltozós idősorokról beszélek, maga az idősor szó alatt az egyváltozós klasszikus idősorokat értem.

1.1.2. Osztályozás

Az osztályozás egyike az adatbányászat/gépi tanulás alap feladatainak. Rendelkezésünkre állnak adatok és címkék. A lehetséges címkék halmaza előre adott, és a címkék kategória típusúak, azaz egymással össze nem hasonlíthatóak, nem létezik közöttük (objektív) részleges sorrendezés se (pl. képről akarunk gyümölcsöket felismerni, ilyenkor lehetnek címkék a következők: alma, körte, szilva). Ez a feltétel független a kategóriákat jelölő szimbólumoktól, ha az előző példában az egyes kategóriákat az 1, 2, 3 szimbólumokkal jelölném, akkor sem mondhatnám azt, hogy a 3-as nagyobb, mint a 2-es, azt pedig főleg nem, hogy 3 = 1 + 2. Adatok alatt bármit el lehet képzelni, kezdve a klasszikus adatrekordoktól az idősorokon át a gráfokig.

Az adatok egy részéhez ismert a címke (célváltozó) értéke, a másik részéhez nem. Az előbbi halmazt nevezzük tanító halmaznak. Az osztályozás feladata, hogy az ismeretlen helyeken találjuk ki a célváltozót, címkézzük fel az ismeretlen adatokat úgy, hogy az eredmény minél közelebb álljon a valósághoz. Ehhez felhasználjuk azokat az adatokat, amelyeknél ismert a célváltozó értéke.

Idősor-osztályozásnál a vizsgált adatok értelemszerűen idősorok.

1.2. Jelölések rendszere

- Θ - egy idősor
- $\Theta[t]$ - az idősor t. eleme
- T - az idősor hossza
- $T[x]$ - az x elemhez tartozó idő érték ($T\left[\Theta[t]\right] = t$)
- N_V - az idősor változóinak a száma
- $CL = \{l_1, l_2, \dots, l_{N_C}\}$ - a lehetséges címkék halmaza
- N_C - osztályok (lehetséges címkék) száma
- $TR = \{\langle \Theta_\mathrm{n}, \mathrm{L_n} \rangle\}_{n=1}^{N_{TR}}$ - a tanítóhalmaz
- $TR[n] = \langle \Theta_\mathrm{n}, \mathrm{L_n} \rangle$ - az n. tanítóminta (idősor - címke pár)
- N_{TR} - tanítóminták száma
- L_n - az n. tanítóminta címkéje, értékét a CL halmazból veszi
- Θ_n - az n. tanítóminta idősora
- $TE = \{\langle \Theta_\mathrm{n}^\mathrm{TE}, \mathrm{L_n^{TE}} \rangle\}_{n=1}^{N_{TE}}$ - a teszthalmaz a kiértékelésekhez, $TE[n]$, $\Theta_\mathrm{n}^\mathrm{TE}$, $\mathrm{L_n^{TE}}$, N_{TE} jelölések hasonlóan a TR-beli megfelelőkhöz
- $\hat{L}_n^{TE}$ - az osztályozó által az n. tesztelő idősorhoz rendelt címke
- $VA = \{\langle \Theta_\mathrm{n}^\mathrm{VA}, \mathrm{L_n^{VA}} \rangle\}_{n=1}^{N_{VA}}$ - a validációs halmaz közbenső kiértékelésekhez, $VA[n]$, $\Theta_\mathrm{n}^\mathrm{VA}$, $\mathrm{L_n^{VA}}$, N_{VA}, $\hat{L}_n^{VA}$ jelölések hasonlóan a TE-beli megfelelőkhöz

1.3. Elterjedt idősor osztályozók

Az idősor-osztályozás problémájára a legegyszerűbb megoldás a már jól bevált osztályozók használata (neurális hálók, logisztikus regresszió, SVM, döntési fák, stb.). Sajnos ezekkel az időbeliséget, a struktúrát nem lehet felhasználni mint információt és az algoritmusok érzékenyek az időbeli eltolásokra. Ráadásul ha az idősorok hossza nem egyezik meg, akkor valamilyen előfeldolgozó lépéssel azonos hosszúságú rekordokká kell az idősorokat transzformálni.

Szintén az egyszerűbb módszerek közé tartoznak a feature extraction módszerek, amikor az idősorokból valamilyen statisztikai információkat nyerünk ki, azokból rekordokat képezünk és ezeket adjuk egy klasszikus osztályozó bemenetére. Pl. minden idősornak vesszük az átlagát, a minimumát, a maximumát és az elemek szórásnégyzetét, majd az így kapott rekordokat SVM-mel osztályozzuk. Ezek a módszerek bizonyos problémákra kifejezetten jól működnek [1]. Fontos viszont, hogy a jó statisztikákat találjuk meg. Az időbeliség információtartalma itt nem feltétlenül veszik el, mivel akár időfüggő statisztikákat is használhatunk (pl. a maximum helye az időtengelyen).

Jelenleg az egyik legelterjedtebb idősor-osztályozási módszer a K legközelebbi szomszéd módszer (K-NN). Ez egy példány alapú módszer, azaz nem építünk modellt az ismert címkéjű adatok tulajdonságai alapján, hanem osztályozási időben megkeressük az osztályozandó adathoz a K leghasonlóbb címkézett adatot, és ezen adatok célváltozóinak értéke alapján döntünk az osztályozandó adat címkéjéről (általában többségi szavazással). A K-NN módszer önmagában nem idősor-osztályozó algoritmus, hanem egy általános módszer, ami bármilyen adat osztályozására alkalmas, ha van egy jól definiált távolságfüggvényünk arra a típusú adatra. Idősor-osztályozásnál egyrészt a klasszikus Euklideszi távolságot használják (ezzel elveszik az időbeliségből származó információtartalom). A másik klasszikus távolságmérték, ami viszont már kifejezetten az idősor-osztályozáshoz lett megalkotva a Dynamic Time Wrapping, azaz a DTW [2]. A DTW távolságok kiszámítása jelentősen lassabb, mint az Euklideszi távolságé. A példány alapú módszerekkel az alapvető baj az, hogy magas a futási idejük, nagyon rosszul skálázódnak, ráadásul osztályozási időben futnak, azaz nem lehet azt megtenni, hogy előre kiszámoljuk a modellt, és utána gyors válaszokat adunk a bejövő kérésekre. Előnyük viszont, hogy már kis méretű tanítóhalmaz mentén is elfogadható eredményeket produkálnak. Ezzel szemben a modell alapú módszerek osztályozási időben elég gyorsak, a számításigényes rész maga a modellépítés, de az külön időben történik. Kevésbé hajlamosak a túltanulásra, mint a példány alapú módszerek, és kellő mennyiségű adat rendelkezésre állása esetén általában jóval pontosabbak. Viszont hátrányuk, hogy egy jó modell felépítéséhez sok tanítómintára van szükség, kis tanítóhalmaz mellett általában sokkal gyengébben teljesítenek, mint a példány alapú módszerek.

Az idősor-osztályozásra gyakran használt, általános, modell alapú módszer a rejtett Markov modellen alapuló osztályozás [3]. Ez a módszer egy olyan modellre épít, hogy feltesszük, hogy a háttérben az állapotok változását egy Markov lánccal tudjuk leírni, és a változók értékeit ezek az állapotok befolyásolják valamilyen valószínűségi modell szerint. A módszer teljesen általános, több problémára is alkalmazható, de jelentős szakértői tudást igényel. A tanuló algoritmusok képesek a Markov lánc állapotátmeneti valószínűségeinek tanulására, illetve a megfigyelési valószínűségek becslésére, sőt, egyes algoritmusok még a Markov lánc struktúráját is finomítani tudják, de minél általánosabb algoritmust használunk, annál több tanítómintára lesz szükségünk a jó eredményhez. A struktúra becslése kifejezetten adatigényes, és a valószínűségek becslése is csökkenti a konvergencia sebességét. Ha viszont megszabjuk a mögöttes struktúrát, akkor nagyon eltérő eredményeket kaphatunk attól függően, hogy a megoldandó problémához mennyire passzolt a kiindulási struktúra. Tehát a

kiindulási struktúra helyes kialakításához magas szintű szakértői tudás szükséges a probléma területén.

Ezen kívül még sok idősor osztályozási módszer létezik. Közülük több olyan van, amik egy-egy speciális idősor-osztályozási problémára lettek kifejlesztve. Azaz egy adott területen nagyon jól teljesítenek, a többi területen viszont akár az alap módszerek is lekörözik őket.

1.4. A kiértékeléshez használt idősor adatbázisok ismertetése

A módszerek hatékonyságának leméréséhez több adatbázist is felhasználtam.

A mérések többségét a világ egyik legnagyobb, publikusan elérhető idősor adatbázisán végeztem [4], amire UCR adatbázis, vagy UCR adatok néven fogok hivatkozni. Ez az adatbázis 20 idősor-osztályozási problémát tartalmaz. A problémák nagyon eltérőek, akár a tanítóminta méretét, akár az osztályok számát, akár az idősorok hosszát tekintjük, így egy megfelelő benchmark adatbázisnak számít. Az adatsoroknál előre kialakított tanító és teszthalmazok vannak, ezen nem változtatok a mérések során, hogy a tesztek eredményei más módszerekkel összehasonlíthatóak legyenek. Az adatbázisra jellemző, hogy az adatsorok nagy részénél a tanítóhalmaz mérete kicsi, ami erősen kedvez a példány alapú módszereknek.

A kiértékeléshez használt másik adatbázis a 2008-as Ford Challenge [5] két adatsora, amikre Ford adatok néven fogok hivatkozni. Mindkét adatsor tanítómintája kellően nagy méretű, így a ShiftTree kiértékeléséhez tökéletesen megfelelnek. A két adatsor hasonló osztályozási problémát ír le: alkatrészeket tesztelnek, változó feszültségeket rájuk kapcsolva, és a kimeneten mérik a feszültséget, ami egy idősort alkot. Ez alapján kell eldönteni, hogy az alkatrészek jók-e, vagy hibásak. A két adatsor abban tér el, hogy az egyik jóval zajosabb. Az adatsorok eredetileg három részre voltak osztva, úgymint tanító-, validáló- és teszthalmaz (az elsőt tették elérhetővé a versenyzők számára a verseny elején, a másodikon mérték vissza a verseny közben az eredményeket, és a harmadikon értékelték ki a végleges beküldött megoldásokat). A tesztekhez az tanító- és a validáló halmazokat összevontam egy nagyobb tanítóhalmazba és az eredeti teszthalmazt használtam én is tesztelésre.

A harmadik ilyen adatbázis a 2007-es KDD TIme Series Challenge [6] 20 adatsora, amikre TSC adatokként fogok hivatkozni. Ezen adatsorok tulajdonságai (méret, hossz, stb.) eléggé hasonlóak az UCR adatokéhoz. Ezt az adatbázist kizárólag vak tesztekre (blind testing) használtam fel. A vak tesztek lényege, hogy az egyes módszerek eredményeit pontosan egyszer mérjük le, így nem tudunk úgy paramétert optimalizálni, hogy közben rátanulunk a teszthalmazra. Természetesen így rosszabb eredmények várhatóak, mint amikor egy adott teszthalmaz mellett pontosan kioptimalizáljuk a paramétereket, de az így kapott modellek jobban megfelelnek a valóságnak, azaz annak, hogy az épített modellek hogyan teljesítenének a gyakorlatban.

A fentiek mellett még két többváltozós adatsort használtam a mérésekhez. Az egyik a 2003-as BCI Competition első adatsora. Ez az adatsor egy 6 változós EEG jelet tartalmaz két osztállyal. Az egyik címke azt jelenti, hogy a kísérleti alany lefele próbálta meg elmozdítani kurzort az EEG-n keresztül az agyhullámaival, azáltal, hogy arra gondolt, hogy a kurzor lefele mozduljon, a másik címke pedig ugyanezt jelenti, csak felfele mozdítással.

A másik többváltozós adatsor digitalizált hanginformációkat tartalmaz [7]. Kilenc különböző férfi mondta ki többször a japán 'ae' magánhangzót, amit felvettek. Ezekből a felvételekből aztán idősorokat állítottak elő. A feladat megállapítani, hogy a minta melyik alanytól származik. Az adatsor érdekessége, hogy eltérő hosszúságú, ugyanakkor rövid idősorokat tartalmaz.

Az egyes adatsorok tulajdonsági láthatóak az 1. táblázatban.

Adatsor neve	Tanítóhalmaz mérete	Teszthalmaz mérete	Osztályok száma	Idősorok hossza	Változók száma
50Words	450	455	50	270	1
Adiac	390	391	37	176	1
Beef	30	30	5	470	1
CBF	30	900	3	128	1
Coffee	28	28	2	286	1
ECG200	100	100	2	96	1
FaceAll	560	1690	14	131	1
FaceFour	24	88	4	350	1
Fish	175	175	7	463	1
GunPoint	50	150	2	150	1
Lighting2	60	61	2	637	1
Lighting7	70	73	7	319	1
OliveOil	30	30	4	570	1
OSULeaf	200	242	6	427	1
SwedishLeaf	500	625	15	128	1
SyntheticControl	300	300	6	60	1
Trace	100	100	4	275	1
TwoPatterns	1000	4000	4	128	1
Wafer	1000	6164	2	152	1
Yoga	300	3000	2	426	1
FordA	3601	1230	2	500	1
FordB	3636	810	2	500	1
TSC01	55	2345	8	1024	1
TSC02	67	1029	2	24	1
TSC03	367	1620	2	512	1
TSC04	178	1085	2	512	1
TSC05	40	1380	4	1639	1
TSC06	155	308	5	1092	1
TSC07	25	995	6	398	1
TSC08	381	760	10	99	1
TSC09	20	601	2	70	1
TSC10	27	953	2	65	1
TSC11	23	1139	2	82	1
TSC12	1000	8236	3	1024	1
TSC13	16	306	4	345	1
TSC14	20	1252	2	84	1
TSC15	467	3840	3	166	1
TSC16	23	861	2	136	1
TSC17	73	936	2	405	1
TSC18	100	550	7	1882	1
TSC19	200	2050	14	131	1
TSC20	267	638	25	270	1
AE	270	370	9	7-29	12
BCI	286	293	2	896	6

1. táblázat: Adatsorok tulajdonságai

1.4.1. Kiértékelési módszerek

Az egyes változatok összehasonlítására alapvetően azt a módszert használtam, hogy az előre leválasztott teszthalmazon mértem le a megfelelő mérőszámokat, és ezeket hasonlítottam össze. Ezt a módszert egyszerű tesztnek nevezem.

Bizonyos esetekben szükség volt arra, hogy a paramétereket optimalizáljam. Ilyenkor értelemszerűen nem használhatom a teszt halmazt a mérésekhez, mivel az torzítaná az eredményeket: a paramétereke így pont úgy lennének beállítva, hogy az adott teszthalmaz számára optimálisak lennének. De ha további adatok állnának rendelkezésre, azokon lefuttatva az osztályozást, feltehetőleg rosszabb értékeket kapnánk a kiértékeléskor. Ezért ha paraméteroptimalizálásra van szükség, akkor a tanítóhalmazon keresztvalidációt használok, azaz a tanítóhalmazt két részre osztom: egy tényleges tanítóhalmazra és egy validációs halmazra. A felosztást úgy csinálom, hogy a különböző osztályok eloszlása hasonló legyen a két halmazban. A tényleges tanítóhalmazon tanítom a modellt, majd a validációs halmazon mérem le a pontosságot. Ezt a mérést több különböző felosztás mellett megismétlem, majd a kapott eredményeket átlagolom, és ezt a mértéket használom az egyes variációk összehasonlításához.

A paraméteroptimalizálás és a módszerek összehasonlítása között található félúton egy algoritmus különböző változatainak összehasonlítása. Amikor ezeket egymással hasonlítom össze, akkor teljes mértékben megfelelnek erre az egyszerű tesztek. Akkor sem követek el komoly kiértékelési hibát, ha a különböző variánsok közül így kiválasztom a legjobbat, és azt hasonlítom össze más algoritmusok eredményeivel. Viszont ilyenkor is megvan az esély - bár sok a sok adatsoron történő tesztelés miatt ez alacsony - hogy az adott variáns csak az adott adatsor halmaz teszthalmazaira optimális, és más teszthalmazokra másik módszer lenne optimális, és ezért az összehasonlításnál a több variánssal rendelkező algoritmus előnybe kerül a többi algoritmussal szemben. Emiatt úgy nevezett vak teszteket is használok, ami azt jelenti, hogy bizonyos adatsorokon minden algoritmusból egyetlen variánst tesztelek és azt is csak egyetlen egyszer, és így hasonlítom össze az algoritmusokat. Azt, hogy melyik variánst használom, más adatsorokon történő egyszerű tesztekkel választom ki. A vak tesztek legbiztosabb fajtája az, amikor a teszthalmazok tényleges címkézése nem áll rendelkezésre, és egy harmadik fél végzi el a kiértékelést. Jelen esetben ez nem így van, mivel rendelkezem a TSC adatok teszthalmazainak címkéivel is. Viszont éppen ezért külön ügyelek arra, hogy ezeken az adatokon már semmilyen modell optimalizációt ne hajtsak végre.

1.4.2. Mérőszámok

A leggyakrabban használt mérőszám a pontosság (accuracy) mérték lesz, ami egy adott problémánál a teszthalmaz helyesen felcímkézett idősorainak a száma, osztva a teszthalmaz méretével, formálisan:

$$Acc = \frac{\sum_{n=1}^{N_{TE}} I\{L_n^{TE} = \hat{L}_n^{TE}\}}{N_{TE}}$$

A pontosság értéke 0 és 1 között van, általában százalékos formában használom. Egy kapcsolódó mérőszám a hibaarány, ami a rosszul címkézett osztályok aránya a teszthalmaz méretéhez, vagy másként fogalmazva a pontosság értéke egyből kivonva:

$$Err = 1 - Acc = \frac{\sum_{n=1}^{N_{TE}} I\{L_n^{TE} \neq \hat{L}_n^{TE}\}}{N_{TE}}$$

Osztályozási feladatoknál sokszor nem csak az a fontos, hogy a pontosság a lehető legnagyobb legyen, hanem az is fontos, hogy bizonyos osztályoknál a hibázás aránya

alacsony legyen és eközben megengedünk nagyobb hibát más osztályoknál. Például egy diagnosztikai rendszer azt mondja valakire, hogy beteg, de valójában nem az, az persze kellemetlen, meg a további vizsgálatok költsége jelentős lehet, de még mindig sokkal jobb, mint ha azt mondaná rá, hogy nem beteg, és később emiatt a páciens meghalna. Látszik, hogy az egyik típusú hibának nagyobb a költsége, mint a másiknak, tehát két azonos pontosságú rendszer között akár jelentős minőségbeli különbség is lehet. Két osztályos problémáknál szokás az egyik osztályt pozitív osztálynak, a másikat pedig negatívnak nevezni[2]. Ilyenkor négy mérőszámot definiálhatunk, ami a 2. táblázatban látható.

		Becsült címke	
		Pozitív	Negatív
Tényleges címke	Pozitív	Igaz pozitív (true positive) $TP = \sum_{n=1}^{N_{TE}} I\{L_P = L_n^{TE} = \hat{L}_n^{TE} = L_P\}$	Hamis negatív (false negative) $FN = \sum_{n=1}^{N_{TE}} I\{L_P = L_n^{TE} \neq \hat{L}_n^{TE} = L_N\}$
	Negatív	Hamis pozitív (false positive) $FP = \sum_{n=1}^{N_{TE}} I\{L_N = L_n^{TE} \neq \hat{L}_n^{TE} = L_P\}$	Igaz negatív (true negative) $TN = \sum_{n=1}^{N_{TE}} I\{L_N = L_n^{TE} = \hat{L}_n^{TE} = L_N\}$

2. táblázat: Két osztályos feladat hibái osztályonként

A fentebb definiált mérőszámok ezen értékek segítségeivel is megadhatóak. Emellett megadható két olyan mérőszám, amelyek szintén az osztályozó minőségét írják le, de az előzőektől eltérően. Az érzékenység (sensitivity) azt mutatja meg, hogy a ténylegesen pozitív entitások közül az osztályozó mennyit címkézett pozitívnak. A fajlagosság (specificity) pedig az osztályozó pontosságát mutatja a ténylegesen negatív entitások között. A fajlagosság helyett sokszor az $1 - fajlagosság$ értéket használják, ami ténylegesen negatív entitások között a hamis pozitívok arányát jelenti. Ezt a mértéket szokás hamis pozitív aránynak is hívni (false positive ratio, FPR). Látható, hogy az érzékenység és a fajlagosság csak egymás kárára javítható adott pontosság mellett.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Err = \frac{FP + FN}{TP + TN + FP + FN}$$

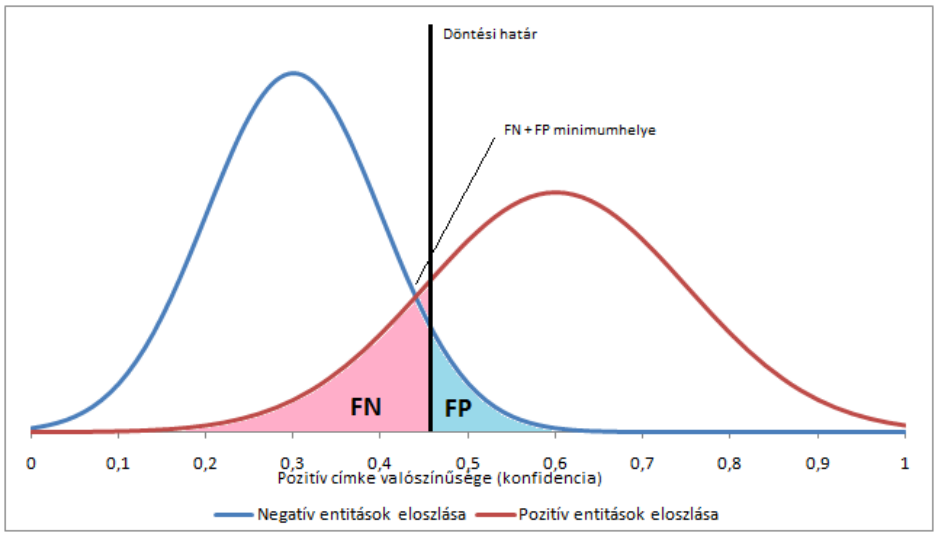
$$Sens = \frac{TP}{TP + FN}$$

$$Spec = \frac{TN}{TN + FP} = 1 - FPR = 1 - \frac{FP}{TN + FP}$$

Bizonyos osztályozók kimenete egy bejövő adatra nem csak egy $\hat{L}$ címke, hanem egy $< \hat{L}, c >$ címke, konfidencia pár. Ezek az osztályozók tehát nem csak azt mondják meg, hogy szerintük az adat melyik osztályhoz tartozik, hanem azt is, hogy mennyire biztosak a dolgukban. Kétosztályos probléma esetén szokás ezt úgy átalakítani, hogy a pozitívnak nevezett osztály esetén megtartjuk a konfidenciát ($c' = c$), a negatív osztály esetén pedig a

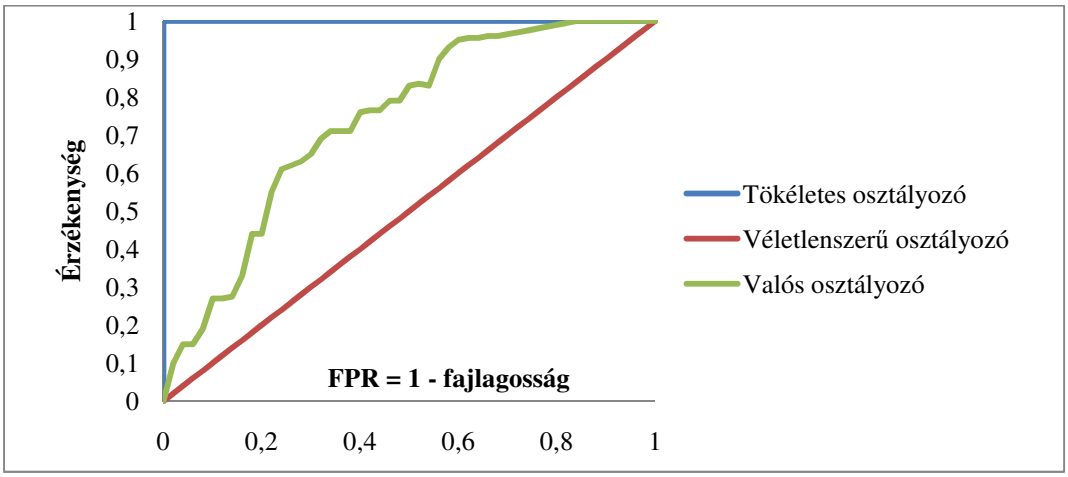
[2] Három- vagy többosztályos problémák is visszavezethetőek erre egy osztályt vagy osztályhalmazt pozitívnak, a többit negatívnak jelölve. A méréseket több különböző osztállyal megismételve a következő mérőszámokat ezekre a problémákra is értelmezhetjük.

konfidenciát kivonjuk egyből: $c' = 1 - c$. Mivel a konfidencia minden esetben 0,5 és 1 közé esik kétosztályos osztályozási problémák esetén[3], így c' jelentése az lesz, hogy mennyire valószínű, hogy az adott adat a pozitív osztályba tartozik. Erre a kimenetre építhetünk egy olyan osztályozót, ami ettől a valószínűségtől függően mondja meg, hogy az adott entitás pozitív-e. Egy adott határ fölött azt mondjuk, hogy a címke pozitív, az alatt pedig azt, hogy negatív. Ennek a határnak a beállításával határozzuk meg a számunkra optimális trade-off-ot az érzékenység és a fajlagosság között, ahogy azt a 1. ábra mutatja.



1. ábra: Döntési probléma kétosztályos esetben

Egy adott döntési határnál az érzékenység és az FPR, egy pontot ad a ROC (Receiver Operating Characteristic) térben. Ez a pont úgy értelmezhető, mint egy trade-off a két mérték között. A határ értékének növelésével mindkét érték monoton növekszik. Ha a határ minden értéke mellett felvesszük ezt a pontot, akkor egy görbét kapunk a ROC térben, amit ROC görbének nevezünk. Ez a görbe a (0,0) pontból indul és az (1,1) pontban végződik. Az osztályozó effajta minőségét egy mérőszámmal a ROC görbe alatti terület nagyságával szokták jellemezni, amit AUC-nak [8] neveznek (Area Under Curve). Példa a ROC görbére a 2. ábra.



2. ábra: Példa ROC görbére

[3] A konfidencia osztályozásnál egy valószínűség érték, hogy mekkora a valószínűsége, hogy az adott adatminta az adott címkéhez tartozik. Az osztályozóban előáll az összes címkéhez a hozzátartozási valószínűség. A kimenetre a legnagyobb ilyen valószínűség kerül, a hozzátartozó címkével. Kétosztályos problémánál ez nyilvánvalóan nem lehet kisebb 0,5-nél.

2. A ShiftTree algoritmus

A ShiftTree [9] egy általam [4] 2008-ban kifejlesztett, modell alapú idősor-osztályozás algoritmus. Ebben a fejezetben bemutatom a ShiftTree mögötti koncepciót, az algoritmus működését és a 2008-as állapotot, amit a munkám során kiindulási alapként felhasználtam.

2.1. Koncepció

A ShiftTree alapötlete az, hogy generáljunk az idősorokhoz dinamikus attribútumokat, és ezeket felhasználva egy klasszikus módszerrel végezzük ez az osztályozást. Az alapötlet lényegi része az, hogy hogyan generáljuk ezeket az attribútumokat, és az, hogy milyen osztályozót használjunk. Az 1.3. fejezetben bemutatott statikus attribútum generálás (statisztika készítés) helyett az attribútumokat dinamikusan, és lokálisan generáljuk. Ehhez a fajta megközelítéshez pedig tökéletesen passzolt a klasszikus (bináris) döntési fák [10] struktúrája, mivel így a korábbi csomópontoktól (korábbi úttól) függően más és más attribútumokat generálhatunk az idősorokhoz, ami alapján a legjobban szétválaszthatjuk a különböző osztályokba tartozó entitásokat. Ezzel az ötlettel az algoritmus egyszerre használja fel az időbeliségben rejlő információkat és egy klasszikus osztályozó megbízhatóságát.

Nulladik lépésként minden idősorhoz rendeljünk hozzá egy-egy szemnek nevezett pointert vagy kurzort (C_i), ami az idősor egy adott elemére mutat (kezdetben mutasson az első elemre). Ezután a dinamikus attribútumok előállítása két lépésben történik. Első lépésben mozgassuk el valahová a szemet az idősoron. Ezt a mozgatást a szemtologató operátorok (EyeShifter Operator, ESO) végzik. Az ESO-k elég sokfélék lehetnek, a lényeg, hogy valamilyen jól meghatározott pontra mozgassák a szemet az időtengely mentén. Fontos megjegyezni, hogy egy ESO két különböző idősornál könnyen eredményezhet különböző szem pozíciókat: pl. ha az ESO a globális maximumra állítja a szemet, és a maximumok pozíciója nem feltétlenül egyezik meg két különböző idősornál. A második lépés az, hogy a szem által mutatott érték, a környezet, az ugrás tulajdonságai, stb. alapján előállítsuk a konkrét attribútum értéket, amit számított értéknek (Calculated Value, CV) nevezünk. Ezt a feladatot egy másik operátorcsalád, a feltételszámító operátorok (Condition Builder Operator, CBO) végzik. A számításhoz a CBO-k több dolgot is felhasználhatnak a szem által mutatott érték mellett, mint például a környezetben található értékeket, a szem időtengelyen való elhelyezkedését, vagy akár az ESO által végrehajtott eltolás tulajdonságait.

A ShiftTree modell (amit néha röviden ShiftTree-ként hivatkozok) tehát egy bináris fa struktúra, aminek minden csomópontja a következő öt tagú struktúrával írható le:

$$Node = \langle ESO_j, CBO_k, TV, PLabel, Child_L, Child_R \rangle$$

Ebben a kifejezésben ESO_j a szem mozgatásához használt ESO, CBO_k pedig a CV kiszámításához használt CBO, azaz az ötös első két tagja mondja meg, hogy a csomópontban hogyan számítjuk ki a dinamikus attribútumot. A $PLabel$ egy olyan függvény, ami minden lehetséges címkéhez egy valószínűség értéket rendel, hogy az adott csomópontban az adott címke mekkora valószínűséggel fordul elő. $Child_L$ és $Child_R$ az aktuális csomópont bal- és jobboldali gyermek csomópontjaira mutató pointerek. A TV egy küszöbérték (Threshold Value), az ennél kisebb dinamikus attribútum értékkel rendelkező idősorok feldolgozása a baloldali, a nagyobbak feldolgozása pedig a jobboldali gyermekcsomópontban folytatódik.

Többváltozós idősorok esetében a fenti struktúra két taggal bővül, amik azt jelzik, hogy mely változón használjuk az operátorok:

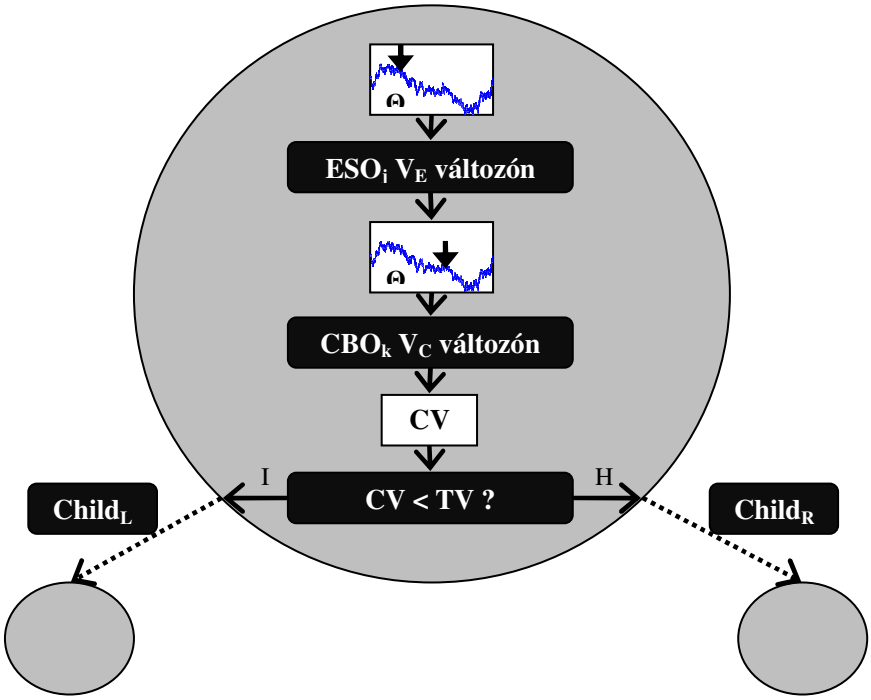
[4] Az igény, és az algoritmus alapötletének első változata konzulensemtől, Gáspár-Papanek Csabától származik. Én ezt az alapötletet dolgoztam át, és így alakult ki a 2.1.-ben ismertetett koncepció.

$$Node = \langle ESO_j, \boldsymbol{V_E}, CBO_k, \boldsymbol{V_C}, PLabel, Child_L, Child_R \rangle$$

V_E annak a változónak az indexe, amin ESO_j-t alkalmazzuk, V_C pedig azt a változót jelöli, amin kiszámoljuk a CV-t CBO_k segítségével. A jelölésből is látszik, hogy a két indexnek nem kell megegyeznie, tehát lehet, hogy úgy számolunk ki egy attribútumot, hogy az első változó maximumhelye környékén vesszük a hatodik változó értékét.

A csomópont szerkezetét mutatja a

3. ábra: a beérkező idősoron először az aktuális ESO végzi a szem pozíciójának beállítását, amihez a V_E változó értékeit használja fel. Ezután az aktuális CBO kiszámít egy számított értéket (CV), amit összehasonlítunk a küszöbértékkel (TV). A feldolgozást az összehasonlítás kimenetétől függően a bal- vagy a jobboldali gyermekcsomópont folytatja. A címkéző- és a tanulóalgoritmus részletes leírása a 2.2. és 2.3. alfejezetekben található.



3. ábra: ShiftTree csomópont szerkezete

2.1.1. ESO-k

Az eredeti ShiftTree algoritmusban az alábbi nyolc féle ESO volt definiálva. Az operátorok a C pozícióban lévő kurzort a C_{new} pozícióba mozgatják a Θ idősoron. Minden ESO-nak az utolsó paramétere az, hogy mely változón hajtjuk végre, de ezt az áttekinthetőség szempontjából elhagyom, ha egyértelmű.

- ESONext(X):
  - $C_{new} = \min(C + X, T)$
  - A szemet X pozícióval előrébb mozdítja, vagy az idősor végére, ha az rövidebb, mint $C + X$.
- ESOPrev(X):
  - $C_{new} = \max(C - X, 1)$
  - A szemet X pozícióval visszább mozdítja, vagy az idősor elejére, a kurzorpozíciótól függően.
- ESOMax(global):
  - $C_{new} = argmax_i(\Theta[i])$
  - A szemet a globális maximumhelyre mozgatja. (Több ilyen esetén az elsőre.)

- ESOMin(global):
  - $C_{new} = argmin_i(\Theta[i])$
  - A szemet a globális minimumhelyre mozgatja. (Több ilyen esetén az elsőre.)
- ESONextMax:
  - $C_{new} = min\{i \mid \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1], i > C\}$
  - A szemet a következő lokális maximumra mozgatja (egyenlőséget megengedünk a követő értékkel). (Ha nincs ilyen, akkor a szem nem mozdul.)
- ESONextMin:
  - $C_{new} = min\{i \mid \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1], i > C\}$
  - A szemet a következő lokális minimumra mozgatja (egyenlőséget megengedünk a követő értékkel). (Ha nincs ilyen, akkor a szem nem mozdul.)
- ESOPrevMax:
  - $C_{new} = max\{i \mid \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1], i < C\}$
  - A szemet az előző lokális maximumra mozgatja (egyenlőséget megengedünk a követő értékkel). (Ha nincs ilyen, akkor a szem nem mozdul.)
- ESOPrevMin:
  - $C_{new} = max\{i \mid \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1], i < C\}$
  - A szemet az előző lokális minimumra mozgatja (egyenlőséget megengedünk a követő értékkel). (Ha nincs ilyen, akkor a szem nem mozdul.)

2.1.2. CBO-k

Az eredeti ShiftTree algoritmusban az alábbi 4 + 2 CBO volt definiálva. A szétbontás oka az, hogy az utolsó kettő CBO speciális típus, úgynevezett feltételállító kiterjesztés (Condition Builder Extension, CBE). A CBE-k olyan CBO-k, amik egy előre megadott CBO-val kiszámítják a CV-t minden változóra, majd ezekből származtatják a tényleges CV-t. Az operátorok a C_{prev} kezdeti és a C aktuális kurzorpozíciókat használják a Θ idősoron. Minden CBO-nak az utolsó paramétere az, hogy mely változón használjuk, de ezt az áttekinthetőség szempontjából elhagyom, ha egyértelmű.

- CBOSimple: $CV = \Theta[C]$
  - A kurzor által mutatott értéket adja vissza.
- CBONormal(μ, σ, L):
  - $CV = average_{i=-L}^{L}\left\{e^{-\frac{(|i|-\mu)^2}{2\sigma^2}}\,\Theta[C+i]\right\}$
  - A kurzor által mutatott érték én annak L sugarú környezetének súlyozott átlagát adja vissza, ahol a súlyok egy (μ, σ) paraméterű normális eloszlásból származnak.
- CBOExp(λ, L):
  - $CV = average_{i=-L}^{L}\{\lambda e^{-\lambda|i|}\,\Theta[C+i]\}$
  - A kurzor által mutatott érték én annak L sugarú környezetének súlyozott átlagát adja vissza, ahol a súlyok egy λ paraméterű exponenciális eloszlásból származnak.
- CBOAvg(L):
  - $CV = average_{i=-L}^{L}\{\Theta[C+i]\}$
  - A kurzor által mutatott érték én annak L sugarú környezetének átlagát adja vissza.
- CBEAverage(CBO_k):
  - $CV = average_{v=1}^{N_V}\{CBO_k(\Theta, v)\}$
  - A CBO_k által az összes változóra kiszámolt értékeket átlagolja.

- CBEVariance(CBO_k):
  - $CV = variance_{v=1}^{N_V}\{CBO_k(\Theta, v)\}$
  - A CBO_k által az összes változóra kiszámolt értékek szórásnégyzetét veszi.

2.2. Osztályozás ShiftTree modellel

A ShiftTree osztályozási folyamata a következő rekurzív függvénnyel írható le (1. algoritmus). A `ShiftTreeLabel` függvény végzi az osztályozást. A függvény bemenete az R csomóponttal reprezentált ShiftTree részfa (kezdetben R a gyökér csomópont), az osztályozandó Θ idősor és az idősorhoz tartozó C szem (kezdetben $C = 1$). A `ShiftCursor`(ESO_j, V_E, Θ, C) függvény a Θ idősor V_E változóján a kezdeti C szem pozíció mellett az ESO_j operátor szerint elmozdítja a szemet. A `CalculateValue`(CBO_k, V_C, Θ, C) függvény a Θ idősor V_C változóján a C szem pozíció mellett az CBO_k operátor szerint kiszámolja a CV értéket.

```
Bemenet: R csomópont, Θ idősor, C szem
Kimenet: L ∈ CL címke
procedure ShiftTreeLabel(R, Θ, C)
1: R → ⟨ESO_j, V_E, CBO_k, V_C, PLabel, Child_L, Child_R⟩
2: if R nem levél then
3:     C_new ← ShiftCursor(ESO_j, V_E, Θ, C)
4:     CV ← CalculateValue(CBO_k, V_C, Θ, C_new)
5:     if CV < TV then
6:         L ← ShiftTreeLabel(Child_L, Θ, C_new)
7:     else
8:         L ← ShiftTreeLabel(Child_R, Θ, C_new)
9:     end if
10:else
11:    L ← argmax_{l_i}(PLabel(l_i)), l_i ∈ [l_1, … , l_{N_L}]
12:end if
13:return L
end procedure
```

1. algoritmus: A ShiftTree címkézési algoritmusa

A címkézési algoritmus tehát annyiból áll, hogy a gyökércsomópontból indulva végighaladunk egy úton a fában amíg el nem érünk egy levél csomópontig. Ott megnézzük, hogy melyik címkének a legnagyobb a valószínűsége, és azt a címkét adjuk vissza. Azt, hogy melyik úton haladunk a fában a kiszámított dinamikus attribútumok értékei határozzák meg. Minden csomópontban alkalmazzuk a megfelelő ESO-t a megfelelő változón, majd kiszámítjuk a CV-t a megfelelő CBO segítségével a megfelelő változón. Utána ezt hasonlítjuk össze az eltárolt TV értékkel: ha a CV értéke kisebb, akkor a baloldali gyermeknek adjuk tovább az idősor feldolgozását, ha nagyobb (vagy egyenlő), akkor a jobboldalinak.

Fontos már itt is észrevenni, hogy a gyökércsomópontban a szem az idősor elejéről indul, de a következő csomópontban már a gyökérben megállapított C_{new} értékkel folytatjuk a feldolgozást. Azaz egy útvonalon végighaladva az addigi ESO-k lánca szabja meg azt, hogy a szem milyen pozícióra mutat.

2.3. A ShiftTree tanítása

A modell tanítása jóval komplexebb, mint a címkézési folyamat, mivel meg kell találnunk, hogy az egyes csomópontokban mely operátorok és változók alkalmazása a legcélravezetőbb. A tanítást szintén egy rekurzív függvénnyel definiáljuk (2. algoritmus), de a valóságban célravezetőbb a kifejtendő csomópontokat egy FIFO sorban eltárolni, és sorban kifejteni őket (szélességi bejárás), mint rekurzívan kifejteni minden gyermeket (mélységi bejárás), főleg a 2.4.1.-ben ismertetett alternatív tanulási módszer miatt.

Bemenet: Egy címkézett idősor halmaz $TR = \{\langle \Theta_n, L_n \rangle\}_{n=1}^{N_{TR}}$ és a hozzá tartozó szemek $Cursors = \{C_n\}_{n=1}^{N_{TR}}$
Kimenet: R csomópont ami a belőle kiinduló részfát reprezentálja
procedure BuildShiftTree$(TR, Cursors)$
1: R üres csomópont létrehozása
2: **for all** $l_i \in CL$ **do**
3: $PLabels(l_i) \leftarrow \frac{|\{n \mid L_n = l_i\}|}{N_{TR}}$
4: **end for**
5: **if** StoppingCriteria$(PLabels) = true$ **then**
6: $R \leftarrow levél$
7: **else**
8: **for all** $ESO_j \in ESO$ **do**
9: **for** $v = 1 \dots N_V$ **do**
10: **for all** $CBO_k \in CBO$ **do**
11: **for** $w = 1 \dots N_V$ **do**
12: **for** $n = 1 \dots N_{TR}$ **do**
13: $C_{new}^{j,v,n} \leftarrow$ ShiftCursor(ESO_j, v, Θ, C_n)
14: $CV_n^{j,v,k,w} \leftarrow$ CalculateValue$(CBO_k, w, \Theta, C_{new}^{j,n,v})$
15: **end for**
16: $[CV^{j,v,k,w,1}, CV^{j,v,k,w,2}, \dots, CV^{j,v,k,w,N_{TR}}] \leftarrow$ Sort$([CV_1^{j,v,k,w}, CV_2^{j,v,k,w}, .., CV_{N_{TR}}^{j,v,k,w}])$
17: **for** $m = 1 \dots N_{TR} - 1$ **do**
18: $TV^{j,v,k,w,m} \leftarrow (CV_m^{j,v,k,w} + CV_{m+1}^{j,v,k,w})/2$
19: $TR_L^{j,v,k,w,m} \leftarrow \{\Theta \mid CV^{j,v,k,w,n} < TV^{j,v,k,w,m}\}$
20: $TR_R^{j,v,k,w,m} \leftarrow \{\Theta \mid CV^{j,v,k,w,n} \geq TV^{j,v,k,w,m}\}$
21: $E^{j,v,k,w,m} \leftarrow$ Entropy$(TR_L^{j,v,k,w,m}, TR_R^{j,v,k,w,m})$
22: **end for**
23: **end for**
24: **end for**
25: **end for**
26: **end for**
27: $\langle j', v', k', w', m' \rangle = argmin_{j,v,k,w,m}(E^{j,v,k,w,m})$
28: **for** $n = 1 \dots N_{TR}$ **do**
29: $C_n \leftarrow C_{new}^{j',v',n}$
30: **end for**
31: $TR_L \leftarrow TR_L^{j',v',k',w',m'}$
32: $TR_R \leftarrow TR_R^{j',v',k',w',m'}$
33: $Cursors_L \leftarrow \{C_n \mid \Theta_n \in \mathrm{TR_L}\}$
34: $Cursors_R \leftarrow \{C_n \mid \Theta_n \in \mathrm{TR_R}\}$
35: $Child_L \leftarrow$ BuildShiftTree$(TR_L, Cursors_L)$
36: $Child_R \leftarrow$ BuildShiftTree$(TR_R, Cursors_R)$
37: $R \leftarrow \langle ESO_{j'}, v', CBO_{k'}, w', TV^{j',v',k',w',m'}, PLabels, Child_L, Child_R \rangle$
38: **end if**
39: **return R**
end procedure

2. algoritmus: A ShiftTree tanulási algoritmusa

Az algoritmus bemenete egy felcímkézett idősor halmaz (tanítóhalmaz) és az ezekhez az idősorokhoz rendelt szemek (kezdetben mindegyik a saját idősorának elejére mutat). A kimenet egy csomópont, ami a belőle induló részfát reprezentálja, tehát az összes rekurzív hívás lefutása után a gyökér csomópont. A StoppingCriteria(*PLabels*) függvény (5. sor) azt ellenőrzi, hogy meg kell-e állnunk, vagy szétbonthatjuk az aktuális csomópontot. A konkrét implementációban ez a függvény akkor ad vissza igaz értéket, ha a csomópont homogén, más szóval, ha a csomópontba kerülő idősoroknak ugyanaz a címkéje, más szóval, ha létezik olyan *i* index, amire $PLabels(l_i) = 1$.

Ha a csomópont nem levél, akkor megkeressük az optimális dinamikus paramétert. Végigmegyünk az összes ESO-CBO kombináción, beleértve azt is, hogy ezeket különböző változókon vizsgáljuk [5] (8-26 sor). Minden egyes dinamikus attribútum jelölt mentén sorrendezzük a tanítóhalmaz idősorait a `Sort` függvény segítségével (16. sor). A rendezés után megnézzük, hogy mennyire lenne jó, ha eszerint az attribútum jelölt szerint az első és a második, a második és a harmadik, ... az utolsó előtti és az utolsó idősor között vágnánk ketté a csomópontot (17-22 sor). A jóság méréséhez a leendő gyermek csomópontok entrópiáinak összegét használjuk, ezt számolja ki az `Entropy` függvény[6] (21. sor):

$$Entropy(TR_L, TR_R) = -\sum_{X\in[L,R]} \frac{|TR_X|}{|TR_L \cup TR_R|} \sum_{i=1}^{N_C} \left(PLabel_X(l_i) log_2 PLabel_X(l_i)\right)$$

Ez megfeleltethető a döntési fáknál klasszikus kiértékelő függvénynek, az információ nyereségnek (information gain), csak a számítások minimalizálása miatt az aktuális csomópont entrópiáját nem számoljuk ki, mivel az minden vágás jelöltnél megegyezik. Azt a vágást választjuk, ahol a gyermek csomópontok entrópiáinak összege minimális (ekkor az információ nyereség maximális). Ha több ilyen van, akkor az első ilyet választjuk.

Ezzel kiválasztottuk az ESO-t, azt a változót, amin az ESO-t alkalmazzuk, a CBO-t, azt a változót, amin a CBO-t alkalmazzuk, illetve azt is, hogy a csomópontba jutott tanítópontokat hogyan osszuk szét a gyermek csomópontok között. A *TV* értéket úgy határozzuk meg, mint két érték átlagát, ahol ez a két érték a kiválasztott dinamikus attribútum mentén rendezett idősorok közül a legnagyobb olyan érték, aminek az idősora a bal gyermekbe kerül és a legkisebb olyan érték, aminek az idősora a jobb gyermekbe kerül. Más szóval a rendezés során a kiválasztott vágási pont által meghatározott két szomszédos idősor dinamikus attribútumának az átlaga adja a küszöbértéket.

Innentől kezdve nincs más dolgunk, mint beállítani a szemeket a kiválasztott ESO szerint, és rekurzívan meghívni a tanítást a bal- és jobboldali gyermek csomópontra. A gyermek csomópontoknál a szemek kiindulópontja a szülőben kiválasztott ESO által beállított pont. Figyeljük meg, hogy a ki nem választott ESO-k nincsenek hatással a szem pozíciójára.

Fontos megjegyezni, hogy a 2. algoritmus csak a tanítás logikáját írja le pusztán demonstrációs célzattal. A gyakorlatban nyilvánvaló, hogy nem érdemes az összes lehetséges tanítóhalmaz felosztást, számított értéket és entrópia értéket eltárolni. Érdemes ehelyett inkrementálisan haladni, azaz eltárolni az eddigi legjobb vágáshoz tartozó változókat, majd sorban kiszámolni a többit. Ha egy kisebb entrópia értéket kapunk valamelyik vágásnál, mint az eddigi optimum, akkor frissítjük a legjobb vágáshoz tartozó változókat. A könnyebb átláthatóság kedvéért ez nem jelenik meg a 2. algoritmusban.

Az alfejezet elején említett FIFO soros megvalósítás úgy működik, hogy amikor a 2. algoritmus 35-36 sorát hajtjuk végre, akkor a leírtak helyett egy FIFO sor végére rakjuk a $<TR_L, Cursors_L>$ struktúrát, majd pedig ugyanezt megtesszük a bal oldali gyermek tanításához szükséges adatokkal, a gyermekekre mutató pointereket pedig üres csomópontokra állítjuk. Miután kiléptünk a `BuildShiftTree` függvényből, egy külső vezérlőmetódus meghívja a `BuildShiftTree` metódust a FIFO sor első elemére, amit el is távolít a sorból, így a félkész fában lévő üres csomópontokat idővel kifejtjük a fa szélességi bejárása szerinti sorban. Nyilván ilyenkor a külső vezérlőmetódus képes csak jelezni, hogy mikor vagyunk kész a tanítással, azaz a teljes fa kifejtésével.

[5] A valódi implementációban bizonyos operátorokat nem kell minden változón kipróbálni, mivel garantáltan ugyanazt eredményezik az összes változón. Ezt az átláthatóság kedvéért a 2. algoritmusban nem jelöltem.
[6] $0log_20$ esetén a 0-ban vett határértéket, azaz 0-t használunk.

2.3.1. Futási idő, skálázhatóság

A címkézési algoritmus, hasonlóan a többi modell alapú osztályozóhoz, kifejezetten gyors, a gyakorlatban emiatt nehezen mérhető az osztályozási idő. Elméletben egy csomópontban a futási időt az operátorok futási ideje határozza meg. A legrosszabb esetben egy ESO futási ideje az idősor hosszával arányos, míg a leglassabb CBO futási ideje az idősor hosszával és a változók számával arányos. Tehát csomópontonként a futási idő $O(N_V T)$, a teljes címkézési idő pedig $O(N_V TD)$, ahol D a fa szintjeinek a száma, a levélszintet kivéve. A gyakorlatban a futási idő ennél várhatóan sokkal rövidebb, mivel (a) az operátorok nagyon ritkán használják fel ténylegesen az egész idősort a számoláshoz, (b) ugyanígy csak néhány operátor használ fel egynél több változót, (c) nem minden idősor kerül a fában a leghosszabb ágra.

A modell építése már jóval hosszabb ideig tart, de ez nem meglepő egy modell alapú algoritmusnál. Egy csomópontban a futási idő $O\left(N_{ESO}N_{CBO}{N_V}^2(N_{TR}T + N_{TR}log_2 N_{TR})\right)$, mivel minden ESO-CBO párt minden lehetséges változópárra megvizsgálunk, és a vizsgálat során minden tanítómintára végrehajtjuk az operátorokat, amiknek a futási ideje a fent tárgyaltak szerint $O(T)$. Ezen kívül minden egyes ilyen vizsgálati ciklusban N_{TR} darab értéket rendezünk, aminek a költsége $O(N_{TR}log_2 N_{TR})$. Mivel az optimális vágás megtalálást a gyakorlatban inkrementális módon célszerű implementálni, a 2. algoritmus 27. sorának $O\left(N_{ESO}N_{CBO}{N_V}^2 N_{TR}\right)$ műveleti igénye nem külön, hanem az egyes vizsgálati ciklusokban jelentkezik, mint egy-egy plusz művelet. Természetesen ez a végeredményen nem változtat.

A bementi halmaz méretében a csomópontonkénti futási idő az $O(N_{TR}log_2 N_{TR})$ redukált alak szerint skálázódik, azaz csomópontonként a futási idő a csomópontba került tanítóminták számának és annak logaritmusának szorzatától függ. A teljes futási időt viszont éppen ezért nagyban befolyásolja az, hogy milyen fa struktúra alakul ki a bemeneti adatokból, mivel két különböző tanítóhalmaz esetén teljesen eltérő lehet az, hogy (a) a fa mennyire "széles", azaz mennyire hasonlít egy teljes bináris fához, (b) a tanítópontok hogyan oszlanak el a gyermek csomópontok között. A gyakorlatban jellemző futási időt a 4.1. alfejezetben mutatom be, összehasonlítva a felgyorsított algoritmusra jellemző futási idővel.

2.4. ShiftTree variánsok

Az eddig ismertetett ShiftTree az alap ShiftTree algoritmus. Ebben az alfejezetben azt a kettő legfontosabb kiterjesztést mutatom be, amikre a munkám hátralevő részében utalni fogok.

2.4.1. Többszörös modellezés

A 2. algoritmusban a kritikus pont az, amikor kiválasztjuk a csomópont legmegfelelőbb konfigurációját a 27. sorban azáltal, hogy minimalizáljuk a gyermek csomópontok entrópiájának összegét. Ilyenkor, ha több olyan konfiguráció is létezik, ami ezt az összeget minimalizálja, akkor (jobb híján) mindig az első ilyet választjuk. Viszont ez nem feltétlenül vezet optimális megoldásra, több olyan példa is mutatható, amikor egy másik konfiguráció választása célravezetőbb lett volna. Erre a problémára kínál megoldást a többszörös modellezés (Multiple Modelling, MM). Az MM módban futó ShiftTree-nél módosítjuk a csomópontok szerkezetét a következőre:

$$Node = \overline{\langle ESO_j, \boldsymbol{V_E}, CBO_k, \boldsymbol{V_C}, PLabel, Child_L, Child_R \rangle}$$

Azaz a csomópont az eddigi hetes (ötös) struktúra helyett egy olyan vektort tartalmaz, aminek az elemei a korábban ismertetett héttagú adatstruktúrák. A tanítás során ahelyett, hogy az első legjobb konfigurációt használnánk, az összes legjobb konfigurációt használjuk egyszerre. Ez természetesen azt is jelenti, hogy a felépült modell nem egy bináris fa lesz, hanem egy olyan struktúra, ahol egy csomópontból több független bináris részfa is indul.

A címkézés úgy zajlik, hogy az osztályozandó idősort egy csomópontba érve továbbküldjük az összes abból kiinduló részfán (természetesen mindig a megfelelő oldali gyermekeket használva). Így a címkézés végére egy címkehalmazunk lesz, amiben megkeressük a leggyakrabban előforduló címkét, és azt rendeljük hozzá a címkézendő idősorhoz.

Ez a megoldás optimálisabb, mint ha erdőt hoznánk létre az ekvivalensen jó fákból, mert az MM során nagy valószínűséggel nincs csomópont többszörözés, mivel egy-egy csomópont több részfának is a gyökere, két különböző részfában pedig valószínűtlen, hogy legyenek egyező csomópontok.

Megmutatható, hogy a módszerrel jelentős pontosságnövekedés érhető el sok esetben. A megoldás fő hátránya, hogy lassú, és nagyon rosszul skálázható. A hátrányok részletesebb ismertetése a 4.2. alfejezetben olvasható.

2.4.2. Egyszerű nyesés

A 2. algoritmus utáni ismertetőben említettem, hogy minden esetben addig építjük a fát, amíg a csomópontba kerülő tanítóhalmaz homogénné nem válik. Az algoritmus fejlesztése során azt tapasztaltam, hogy bármilyen más szintbeli vagy a pontba eljutó tanítóhalmaz számosságára vagy a címkék arányára vonatkozó leállási feltétel összességben csak ront az eredményeken.

A teljes kifejtés egyik hátránya viszont az, hogy könnyen előfordulhatnak olyan csomópontok, amibe csak néhány tanítópont jut el. Ezeknek a csomópontoknak a kiválasztott konfigurációja viszont nagy eséllyel nem jellemzi a különböző címkék közötti eltéréseket, mivel az operátorok/változók kiválasztását csak néhány példa alapján végzi el. De néhány még ezek közül a csomópontok közül is fontos tulajdonságát írja le az adathalmaznak, így elhagyásuk ront a modell pontosságán, ugyanakkor nem tudjuk előre, hogy melyek lesznek a fontos, és melyek a felesleges kis méretű csomópontok. Ráadásul a felesleges csomópontok nem csak a modell méretét növelik, hanem olyan jelenségekre is könnyen rátanulhatnak, amik nem az egyes osztályokba tartozó mintákat jellemzik, hanem csak a tanítóhalmazt, vagy esetleg a példákra rakódott zajt. Ezt a jelenséget nevezik túltanulásnak (overfitting).

A problémára egy egyszerű megoldás egy egyszerű nyesési eljárás alkalmazása. A nyesési eljárásoknak két fajtája létezik, az elő- és az utónyesés. Az előbbi a tanítás közben fut, és annak során mondja meg egy adott csomópontról, hogy azt érdemes-e kifejteni, vagy inkább nyessük le a belőle kiinduló részfát. Az előnyesés egyfajta komplex leállási feltételként is felfogható. Az utónyesés egy már teljesen felépített modellen fut, és utólag vizsgáljuk meg az egyes csomópontokat, és döntjük el, hogy a belőlük kiinduló részfát lenyessük-e, vagy hagyjuk érintetlenül.

Az egyszerű nyesés egy utónyesési eljárás, így egy teljesen kifejtett ShiftTree-ből idul. A lényege, hogy veszünk egy idősor halmazt, amihez megkeressük a legjobban illeszkedő, a gyökérből induló részfát a modellünkben. Azokat a csomópontokat, melyek nem elemi ennek a részfának, lenyessük a fáról. A legjobb illeszkedést a pontossággal definiáljuk, azaz egy nem levél csomópontot akkor minősítünk levéllé, ha a csomópontba eljutott idősorok közül maga a csomópont többet osztályoz helyesen, mint a belőle kiinduló részfa.

Ezt a nyesési eljárást az MM modellen két lépésben is használhatjuk, ahol a második lépés opcionális. Egyrészt a nyesés segítségével a többszörös modellből kiválaszthatjuk az idősor halmazra legjobban illeszkedő egyszerű modellt (fát). Másrészt ezt a fát továbbnyeshetjük az előző bekezdésben leírtak szerint.

A módszer hatékonysága nagyban függ attól a halmaztól, amihez a legjobban illeszkedő részfát keressük. Ha sikerül olyan halmazt találnunk, ami eléggé eltér a tanítóhalmaztól, de

mégis ugyanazt a problémát írja le, ráadásul a tanítóhalmaz méretével összevethető, akkor az egyszerű nyesés alkalmazásával sokat lehet javítani a modellen. Sajnos a gyakorlatban annyit tehetünk, hogy a tanítóhalmazból próbáljuk leválasztani ezt a validáló halmazt, amit utána az egyszerű nyeséshez használunk. Mivel a tanítóhalmazok mérete általában kicsi, ezért a kettévágásuk után sokszor mindkét létrejövő halmaz túl kicsi lesz, így már a kezdeti modellünk is nagyon zajos, és az egyszerű nyesés is csak korlátozottan lesz használható. Tehát az egyszerű nyesés csak abban az esetben használható eredményesen, amikor a tanítóhalmaz elég nagy ahhoz, hogy le lehessen belőle választani egy validáló halmazt, de még ilyenkor sem garantált, hogy sikerül olyan halmazt leválasztanunk, ami az egyszerű nyesés segítségével növelni tudja a modell pontosságát.

A módszer egy másik felhasználási módja az, ha a teszthalmazt használjuk az egyszerű nyeséshez, így megkapjuk azt a részfát, amire a teszthalmaz a legjobban illeszkedik. Az illeszkedés mértéke (pontosság) természetesen nem a modellünk valódi pontossága lesz, de jól közelíti azt a maximális pontosság értéket, amit az adott operátorkészlettel a ShiftTree-vel el lehet érni. Az így kapott felső pontosság korlátot ezután az elemzési folyamatban ellenőrzésre és az egyes ShiftTree módszerek egyfajta összehasonlítására lehet használni.

2.5. Az algoritmus hatékonyságáról

Megmutatható, hogy a ShiftTree algoritmus pontossága, csak a 2.1. alfejezetben ismertetett egyszerű operátorokat használva, az esetek többségében meggyőzően jobb, mint azoké a klasszikus (nem idősor) osztályozóké, amiket idős-osztályozásra használnak. A pontosság megközelíti a példány alapú módszerek pontosságát, de egy kevéssel elmarad tőlük. A tanítás sebessége elfogadható (lásd 4.1. fejezet).

A pontosság tovább növelhető egy kicsivel, ha az MM módszert alkalmazzuk, de a futási idő növekedése sokkal nagyobb mértékű, mint a pontosság növekedése, ráadásul a modell mérete is a sokszorosára nő. Ha sikerül az egyszerű nyesést jól alkalmazni, akkor ez utóbbi probléma kikerülhető, és a pontosság tovább nőhet egy kicsivel. A pontos értékek a 4. fejezetben találhatóak, ahol az alap algoritmust összehasonlítom a fejlesztett változatokkal.

A pontosság mellett másfajta hatékonyságokat is érdemes megvizsgálni. Egyrészt a címkézés ideje jóval gyorsabb, mint a példány alapú módszereknél, így a kicsivel alacsonyabb pontosság is elfogadhatóvá válik, ha a címkézés sebességére is vannak követelmények (és a legtöbb gyakorlati problémában vannak ilyen követelmények).

Másrészt a gyakorlatban fontos az is, hogy a felépített modellek értelmezhetőek legyenek, fontos, hogy emberek számára is értelmezhető legyen, hogy az adott idősor miért kapta azt a címkét, amit. A modell értelmezhetőségének egy másik fontos aspektusa az, hogy ha egy modell értelmezhető, akkor a felépített modellből következtethetünk az adatok tulajdonságaira, jelen esetben az egyes osztályokba tartozó idősorok milyenségére. Például ha a FORD adatsorokra épített modell értelmezhető, akkor magát az alkatrészt ismerve akár az is kideríthető, hogy hol van a hiba pontos helye. A ShiftTree által épített modellek értelmezhetősége a használt operátorok értelmezhetőségétől függ. Mivel az operátorok elég egyszerűek, ezért a modellek értelmezése könnyű (megfelelő szakértői tudás mellett).

Harmadrészt a ShiftTree egyik nagy előnye az általánosság melletti bővíthetőség. Azaz a már definiált egyszerű operátorokkal is kielégítő pontosság érhető el teljesen különböző területeken, szakértői tudás bevonása nélkül, viszont a terület szakértője bármikor definiálhat olyan, szakértői tudásra támaszkodó, operátorokat, amikkel az adott területen az algoritmus hatékonyabbá tehető. Ez a kettősség, az idősor-osztályozás területén, egyedülálló képessége az algoritmusnak.

3. Új operátorok

Ebben a fejezetben bemutatom azokat az új operátorokat, amikkel kibővítettem a ShiftTree alapvető operátorkészletét. Az új ESO-k és CBO-k mellett egy teljesen új operátorcsaládot is létrehoztam, a virtuális változó operátorok családját. Ezeket is ismertetem a fejezet végén.

3.1. ESO-k leírása

A létrehozott ESO-kat két részre bontottam attól függően, hogy a már meglévőek továbbfejlesztésével hoztam-e őket létre, vagy sem.

3.1.1. Fejlesztett ESO-k

Az alábbiakban bemutatom azokat az ESO-kat, amiket továbbfejlesztettem.

- ESONext(*X*[%]):
  - A fejlesztés lényege, hogy most már nem csak egy fix ugrástávolság adható meg, hanem megadható az ugrás mértéke az idősor hosszának függvényében is. Ez olyan esetekben lehet célszerű, amikor egy problémán belül az idősorok hossza eltér, vagy amikor több problémát akarunk ugyanazzal az operátorkészlettel kezelni, ahol az egyes problémáknál eltérnek a hosszak.
  - normál mód: $C_{new} = C + X$
  - százalékos mód: $C_{new} = C + \left\lfloor \frac{X}{T} \right\rfloor$
- ESOPrev(*X*[%]):
  - Az előzőhöz hasonlóan itt is lehetővé tettem az ugrás méretének százalékos megadást az idősor hosszának függvényében.
  - normál mód: $C_{new} = C - X$
  - százalékos mód: $C_{new} = C - \left\lfloor \frac{X}{T} \right\rfloor$
- ESOMax(global | sofar):
  - Az eddigi módot elneveztem globális módnak. Emellett egy másik mód is használható, ami a szem aktuális pozíciójáig veszi csak figyelembe az idősort, így a szemig tartó részen keresi a maximális értéket. Ez a változat különösen fontos lehet kombinált operátorokban.
  - sofar mód: $C_{new} = argmax_{i=1\ldots C}(\Theta[i])$
  - global mód: $C_{new} = argmax_{i=1\ldots T}(\Theta[i])$
- ESOMin(global | sofar):
  - Az előzővel analóg fejlesztés, a minimum érték kereshető a szem aktuális pozíciójáig vizsgálva az idősort, illetve globálisan is.
  - sofar mód: $C_{new} = argmin_{i=1\ldots C}(\Theta[i])$
  - global mód: $C_{new} = argmin_{i=1\ldots T}(\Theta[i])$
- ESONextMax(*X*):
  - $C_{new} = \begin{pmatrix} i \mid \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1], i > C, \\ \sum_{k=C+1}^{T-1} I_{\{\Theta[k]>\Theta[k-1],\Theta[k]\geq\Theta[k+1]\}} = X - 1 \end{pmatrix}$
  - Megadható, hogy a szemtől számítva hányadik lokális maximumra mozgassa a szemet. Ha nincs ilyen lokális maximum, akkor az idősor vége felé lévő irányban a szem eredeti pozíciójától legtávolabb lévő lokális maximumra ugrik. Ha ilyen sincs, akkor nem mozdítja a szemet. Fontos lehet olyan idősoroknál, ahol a lokális maximumok érdekesek, de elég sok van belőlük.

- ESONextMin(*X*):
  - $C_{new} = \begin{pmatrix} i \mid \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1], i > C, \\ \sum_{k=C+1}^{T-1} I_{\{\Theta[k]<\Theta[k-1],\Theta[k]\leq\Theta[k+1]\}} = X - 1 \end{pmatrix}$
  - Az előzőhöz hasonlóan az idősor végének irányában a szemtől az *X*. lokális minimumra mozdítja a szemet.
- ESOPrevMax(*X*):
  - $C_{new} = \begin{pmatrix} i \mid \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1], i < C, \\ \sum_{k=1}^{C-1} I_{\{\Theta[k]>\Theta[k-1],\Theta[k]\geq\Theta[k+1]\}} = X - 1 \end{pmatrix}$
  - Az előzőhöz hasonlóan az idősor elejének irányában a szemtől az *X*. lokális maximumra mozdítja a szemet.
- ESOPrevMin(*X*):
  - $C_{new} = \begin{pmatrix} i \mid \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1], i < C, \\ \sum_{k=1}^{C-1} I_{\{\Theta[k]<\Theta[k-1],\Theta[k]\leq\Theta[k+1]\}} = X - 1 \end{pmatrix}$
  - Az előzőhöz hasonlóan az idősor elejének irányában a szemtől az *X*. lokális minimumra mozdítja a szemet.

3.1.2. Új ESO-k

Az alábbiakban bemutatom azokat az ESO-kat, amelyeket nem a korábbi operátorok fejlesztéseként hoztam létre. A jelölések megegyeznek a 2.1.1. jelöléseivel.

- ESOClosestMax:
  - $C_{new} = min_{|C-i|}\{i \mid \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1]\}$
  - A szem jelenlegi pozíciójához legközelebb eső lokális maximumra mozgatja a szemet. Ha nincs ilyen, akkor a szemet a helyén hagyja.
- EOSClosestMin:
  - $C_{new} = min_{|C-i|}\{i \mid \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1]\}$
  - A szem jelenlegi pozíciójához legközelebb eső lokális minimumra mozgatja a szemet. Ha nincs ilyen, akkor a szemet a helyén hagyja.
- ESOGreaterMax:
  - $C_{new} = argmax(\Theta[ESONextMax(1)], \Theta[ESOPrevMax(1)])$
  - A szem pozíciójához előre és visszafelé irányban lévő legközelebbi lokális maximumok közül ahhoz mozgatja a szemet, amelyik értéke nagyobb. Fontos lehet olyan adatoknál, ahol az egyes osztályok között egy kis késleltetés a különbség.
- ESOGreaterMin:
  - $C_{new} = argmax(\Theta[ESONextMin(1)], \Theta[ESOPrevMin(1)])$
  - Az előzőhöz hasonló operátor. A szem pozíciójához előre és visszafelé irányban lévő legközelebbi lokális minimumok közül ahhoz mozgatja a szemet, amelyik értéke nagyobb.
- ESOLesserMax:
  - $C_{new} = argmin(\Theta[ESONextMax(1)], \Theta[ESOPrevMax(1)])$
  - Az előzőhöz hasonló operátor. A szem pozíciójához előre és visszafelé irányban lévő legközelebbi lokális maximumok közül ahhoz mozgatja a szemet, amelyik értéke kisebb.
- ESOLesserMin:
  - $C_{new} = argmin(\Theta[ESONextMin(1)], \Theta[ESOPrevMin(1)])$
  - Az előzőhöz hasonló operátor. A szem pozíciójához előre és visszafelé irányban lévő legközelebbi lokális minimumok közül ahhoz mozgatja a szemet, amelyik értéke kisebb.

- ESOMaxInNextInterval(X):
  - $C_{new} = argmax_{i=0\ldots X}(\Theta[C+i])$
  - A szem pozíciójától előre irányban X távolságon belül a legnagyobb értékre ugrik. Fontos lehet hosszabb idősoroknál, ahol a globális maximum kevés információt hordoz az egyes osztályokról.
- ESOMinInNextInterVal(X):
  - $C_{new} = argmin_{i=0\ldots X}(\Theta[C+i])$
  - Az előzőhöz hasonló operátor. A szem pozíciójától előre irányban X távolságon belül a legkisebb értékre ugrik.
- ESOMaxInPrevInterval(X):
  - $C_{new} = argmax_{i=0\ldots X}(\Theta[C-i])$
  - Az előzőhöz hasonló operátor. A szem pozíciójától vissza irányban X távolságon belül a legnagyobb értékre ugrik.
- ESOMinInPrevInterval(X):
  - $C_{new} = argmin_{i=0\ldots X}(\Theta[C-i])$
  - Az előzőhöz hasonló operátor. A szem pozíciójától vissza irányban X távolságon belül a legkisebb értékre ugrik.
- ESOBegin:
  - $C_{new} = 1$
  - Az idősor elejére ugrik vissza.
- ESOEnd:
  - $C_{new} = T$
  - Az idősor végére ugrik.
- ESOStay:
  - $C_{new} = C$
  - Nem mozdítja el a szemet. Ha egy kitűntetett pontban több számított érték is jól használható attribútumot ad, akkor fontos lehet.
- ESOClosestLocal:
  - $C_{new} = argmin\begin{pmatrix}|\mathrm{C} - ESONextMax(1)|, |\mathrm{C} - ESONextMin(1)|,\\ |\mathrm{C} - ESOPrevMax(1)|, |\mathrm{C} - ESOPrevMin(1)|\end{pmatrix}$
  - A legközelebbi lokális szélsőértékre ugrik.
- ESOGreaterDistLocal:
  - $C_{new} = argmax\begin{pmatrix}|\Theta[C] - \Theta[ESONextMax(1)]|,\\ |\Theta[C] - \Theta[ESONextMin(1)]|,\\ |\Theta[C] - \Theta[ESOPrevMax(1)]|,\\ |\Theta[C] - \Theta[ESOPrevMin(1)]|\end{pmatrix}$
  - Az aktuális értéktől abszolút értékben a leginkább eltérő, a pont közvetlen környezetében lévő lokális szélsőértékre ugrik.
- ESOLesserDistLocal:
  - $C_{new} = argmin\begin{pmatrix}|\Theta[C] - \Theta[ESONextMax(1)]|,\\ |\Theta[C] - \Theta[ESONextMin(1)]|,\\ |\Theta[C] - \Theta[ESOPrevMax(1)]|,\\ |\Theta[C] - \Theta[ESOPrevMin(1)]|\end{pmatrix}$
  - Az aktuális értéktől abszolút értékben a legkevésbé eltérő, a pont közvetlen környezetében lévő lokális szélsőértékre ugrik.

- ComplexESO($\overrightarrow{ESOs}$):
  - $C_{new} = ShiftCursor(\overrightarrow{ESOs}[S], v, \Theta, \dots\ ShiftCursor(\overrightarrow{ESOs}[1], v, \Theta, \mathrm{C}) \dots)$
  - Sorban végrehajtja a paraméterként megadott ESO-kat a szemen. Azaz elmozdítja a szemet az első ESO szerint, majd az így kapott pozícióból elmozdítja a második ESO szerint és így tovább. Fontos, hogy a paraméterként kapott ESO-k ugyanazt a *v* indexű változót használják. Ezzel az összetett operátorral bonyolultabb szem tologató műveletek is megvalósíthatóvá válnak.

3.2. Új CBO-k leírása

A 2.1.2.-ben definiált CBO-k továbbfejlesztésére nem volt szükség, ezért az ESO-któl eltérően itt csak teljesen új CBO-kat mutatok be alább. A jelölések megegyeznek a 2.1.2.-ben használtakkal.

- CBOLinear(L):
  - $CV = average_{i=-L}^{L}\left\{\frac{1}{\min\{|i|,1\}}\Theta[C+i]\right\}$
  - A kurzor által mutatott érték én annak L sugarú környezetének súlyozott átlagát adja vissza. A súlyok a szem pozíciójától való távolságok reciprokai. A szem által mutatott érték súlya 1.
- CBODeltaT(norm | abs):
  - norm mód: $CV = C - C_{prev}$
  - abs mód: $CV = \left|C - C_{prev}\right|$
  - Az ESO által kiváltott ugrás hosszát (kszem pozíció és előző szem pozíció különbsége), vagy annak abszolút értéket adja vissza a megadott módtól függően. Ez jelenleg az egyetlen olyan operátor, ami tisztán az időtengelyen dolgozik, és figyelmen kívül hagyja a változók értékeit.
- CBOTimeSensitive(norm | abs):
  - norm mód: $CV = \frac{\Theta[C]}{C - C_{prev}}$
  - abs mód: $CV = CV = \frac{\Theta[C]}{|C - C_{prev}|}$
  - A szem által mutatott érték és az ugrás hosszának (vagy abszolút hosszának) hányadosát adja vissza. Amennyiben az ugrás hossza 0 volt, 1-gyel oszt.
- CBOAverage(sofar | delta):
  - sofar mód: $CV = average_{i=1\dots C}\{\Theta[i]\}$
  - delta mód: $CV = average_{i=C_{prev}\dots C}\{\Theta[i]\}$
  - Egy adott intervallum összes értékének átlagát adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van. A sofar mód által visszaadott érték a szem pozíciójára jellemző érték, míg a delta módban az ugrásra jellemző értéket kapunk.
- CBOVariance(sofar | delta):
  - sofar mód: $CV = variance_{i=1\dots C}\{\Theta[i]\}$
  - delta mód: $CV = variance_{i=C_{prev}\dots C}\{\Theta[i]\}$
  - Az előző operátorhoz hasonló. Egy adott intervallum összes értékének szórásnégyzetét adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van.

- CBOMaxAvg(sofar | delta):
  - sofar mód: $CV = average_{i=1\ldots C}\{\Theta[i] | \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1]\}$
  - delta mód: $CV = average_{i=C_{prev}\ldots C}\{\Theta[i] | \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1]\}$
  - Egy adott intervallumban előforduló lokális maximumok átlagát adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van. A lokális szélsőértékek sok idősor osztályozási problémánál fontos adatok, ezt próbálja meg kihasználni ez az operátor.
- CBOMinAvg(sofar | delta):
  - sofar mód: $CV = average_{i=1\ldots C}\{\Theta[i] | \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1]\}$
  - delta mód: $CV = average_{i=C_{prev}\ldots C}\{\Theta[i] | \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1]\}$
  - Az előzőhöz hasonló operátor. Egy adott intervallumban előforduló lokális minimumok átlagát adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van.
- CBOMaxVar(sofar | delta):
  - sofar mód: $CV = variance_{i=1\ldots C}\{\Theta[i] | \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1]\}$
  - delta mód: $CV = variance_{i=C_{prev}\ldots C}\{\Theta[i] | \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1]\}$
  - Az előzőhöz hasonló operátor. Egy adott intervallumban előforduló lokális maximumok szórásnégyzetét adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van.
- CBOMinVar(sofar | delta):
  - sofar mód: $CV = variance_{i=1\ldots C}\{\Theta[i] | \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1]\}$
  - delta mód: $CV = variance_{i=C_{prev}\ldots C}\{\Theta[i] | \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1]\}$
  - Az előzőhöz hasonló operátor. Egy adott intervallumban előforduló lokális minimumok szórásnégyzetét adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van.
- CBOMaxCount(sofar | delta):
  - sofar mód: $CV = \sum_{i=1}^{C} I_{\{\Theta[i] | \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1]\}}$
  - delta mód: $CV = \sum_{i=C_{prev}}^{C} I_{\{\Theta[i] | \Theta[i] > \Theta[i-1], \Theta[i] \geq \Theta[i+1]\}}$
  - Az előzőhöz hasonló operátor. Egy adott intervallumban előforduló lokális maximumok számát adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van.
- CBOMinCount(sofar | delta):
  - sofar mód: $CV = \sum_{i=1}^{C} I_{\{\Theta[i] | \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1]\}}$
  - delta mód: $CV = \sum_{i=C_{prev}}^{C} I_{\{\Theta[i] | \Theta[i] < \Theta[i-1], \Theta[i] \leq \Theta[i+1]\}}$
  - Az előzőhöz hasonló operátor. Egy adott intervallumban előforduló lokális minimumok számát adja vissza. Ez az intervallum módtól függően vagy az idősor elejétől a szem pozíciójáig tart, vagy az szem előző és jelenlegi pozíciója között van.
- CBOMedian(*B*, *F*):
  - $CV = median_{i=-B\ldots C}\{\Theta[i]\}$
  - A szem pozíciójától vissza irányban *B*, előre irányban *F* hosszú intervallumból a medián érétkét adja vissza.

3.3. A virtuális változó operátorok családja

A virtuális változó operátorok (Virtual Variable Operators, VVO) családja egy új operátor család a ShiftTree algoritmusban. A korábban bemutatott operátorcsaládoktól eltérően ezek az operátorok nem vesznek részt közvetlenül a dinamikus attribútumok kialakításában, a tevékenységük leginkább az előfeldolgozáshoz hasonlítható.

A VVO-k feladata az, hogy egy vagy több változót felhasználva új változókat hozzanak létre az idősorokhoz. Erre több esetben is szükség lehet, mivel sok esetben az egyes változók önmagukban kevesebb információt hordoznak, mint valamilyen kombinációjuk. Bár az alap algoritmus is képes az operátoraival bizonyos szinten együtt kezelni egy idősor több változóját, és az ilyen jellegű információkat képes korlátozott mértékben felhasználni, a VVO-k segítségével még több változók közötti információhoz férhetünk hozzá.

A konkrét operátor típusok leírása előtt érdemes még három témát érinteni, hogy megértsük, hogy a VVO-k pontosan hogyan kapcsolódnak be a ShiftTree által nyújtott struktúrába. Az első ilyen téma a CBE-k és a VVO-k viszonya. A VVO-k abban térnek el a CBE-ktől, hogy nem a CBO-k által számított értékeknek veszik valamilyen kombinációját, hanem közvetlenül a változókon dolgoznak, és az így előállított új változón dolgoznak majd mind az ESO-k, mind a CBO-k. Azaz egyrészt a dinamikus attribútumok számításánál a műveletek sorrendje fordított (először kombinálás, majd CBO-val *CV* számítása), másrészt a szem pozícióját a VVO által előállított virtuális változó alapján is meghatározhatjuk az ESO-k segítségvel.

A második annak a tisztázása, hogy mit jelent a virtuális változó. Gyakorlati szempontból nem sokban tér el a hagyományos változókból, mivel a CBO-k és ESO-k számításai miatt legtöbbször nem érdemes ténylegesen virtuális változókat létrehozni, amiknek az értékét mindig akkor számítjuk ki, amikor szükség van rájuk, mivel ezzel feleslegesen megnövelnénk a számítási igényt. Érdemesebb tehát a virtuális változókat előre kiszámolni, és hozzáadni az idősor eredeti változóihoz, még úgy is, hogy ezzel megnöveljük a memóriafoglalást. Viszont ezek a változók elkülönülnek abból a szempontból, hogy további kombinálásuk akár az eredeti, akár más virtuális változókkal értelmetlen, vagy legalábbis erősen kérdéses a dolog értelme, mivel ilyenkor néhány eredeti attribútumot a dinamikus attribútum kiszámításakor többször többféle módon is felhasználnánk, sokszor eléggé nehezen követhető módon. Ez ráadásul könnyen alapja lehetne az operátor alapú túltanulásnak, amikor az egyes operátorok annyira bonyolulttá válnak, hogy túlságosan pontosan leírják a tanítóhalmazt, vagy annak egyes részeit, de a felépített modell mégis pontatlan lesz a teszthalmazon. Éppen ezért a virtuális változók nem lehetnek alanyai a CBE-knek, illetve esetlegesen a jövőben bevezetett kombinációs operátoroknak. Az viszont lehetséges, hogy az előfeldolgozási szakaszban a VVO-k segítségével virtuális változókból újabb virtuális változókat hozzunk létre, amennyiben erre szükség van. Így minden VVO tekinthető összetett operátornak is, amennyiben más VVO kimenetét kapja bemenetként.

A harmadik téma a modellek értelmezhetősége. Itt ugyanazt lehet elmondani, mint az ESO-k és CBO-k kapcsán: ameddig a VVO-k kellően egyszerűek, vagy a szakértő által kialakítottak, addig a segítségükkel felépített modell is értelmezhető marad. Az értelmezhetőség megtartása a másik oka annak, hogy a virtuális változók nem vehetnek részt a legtöbb kombinációban.

3.3.1. VVO-k leírása

A VVO-k bemenete egy vagy több változó, amit a változó indexével v_i -vel jelölök. Amennyiben a VVO bármennyi változóval képes dolgozni, a bemenet egy vektor ($\vec{V} = [v_i]_{i=1}^{V}$), ami változó indexeket tartalmaz. A Θ idősor v_i indexű változóját a Θ_{v_i} értéksorral jelölöm. A kimenetük egy virtuális változó Θ_{vv}, ami egy értéksor, aminek hossza megegyezik a Θ idősor hosszával. Az operátorok leírásánál az értéksorokon végzett műveletek tagonként

értelmezendőek és az eredményük egy másik értéksor. (azaz $\Theta_{vv} = \Theta_{v_i} + \Theta_{v_j} \Leftrightarrow \Theta_{\mathrm{vv}}[\mathrm{t}] = \Theta_{\mathrm{v_i}}[\mathrm{t}] + \Theta_{\mathrm{v_j}}[\mathrm{t}],\ \mathrm{t} = 1 \ldots \mathrm{T}$). A jelenlegi implementációban az alábbi tíz VVO szerepel:

- VVOSub(norm | abs, v_i, v_j):
  - norm mód: $\Theta_{vv} = \Theta_{v_i} - \Theta_{v_j}$
  - abs mód: $\Theta_{vv} = \left|\Theta_{v_i} - \Theta_{v_j}\right|$
  - A visszaadott virtuális változó a bemeneti változók tagonkénti különbségeként (vagy paramétertől függően abszolút különbségeként) előálló értéksor.
- VVODiv(v_i, v_j):
  - $\Theta_{vv} = \frac{\Theta_{v_i}}{\Theta_{v_j}}$
  - Az előállított virtuális változó a bemeneti változók tagonkénti hányadosaiként előálló értéksor. A nullával való osztás elkerülendő, ha $\Theta_{v_j}[t]$ nulla valamely *t*-re, akkor $\Theta_{vv}[t] = 10^{100}$, vagy valami hasonlóan nagy érték, ami a gyakorlatban máshogyan nem fordulhat elő.
- VVOAvg($\vec{V}$):
  - $\Theta_{vv} = average_{i=1\ldots V}\left\{\Theta_{\vec{V}[i]}\right\}$
  - A virtuális változó a bemeneti változók tagonkénti átlagai által előállított értéksor.
- VVOVar($\vec{V}$):
  - $\Theta_{vv} = variance_{i=1\ldots V}\left\{\Theta_{\vec{V}[i]}\right\}$
  - A virtuális változó a bemeneti változók tagonkénti szórásnégyzetei által előállított értéksor.
- VVOMax($\vec{V}$):
  - $\Theta_{vv} = maximum_{i=1\ldots V}\left\{\Theta_{\vec{V}[i]}\right\}$
  - A virtuális változó a bemeneti változók tagonkénti maximumai által előállított értéksor.
- VVOMin($\vec{V}$):
  - $\Theta_{vv} = minimum_{i=1\ldots V}\left\{\Theta_{\vec{V}[i]}\right\}$
  - A virtuális változó a bemeneti változók tagonkénti minimumai által előállított értéksor.
- VVOLength($\vec{V}$):
  - $\Theta_{vv} = \sqrt{\sum_{i=1}^{V} \Theta_{\vec{V}[i]}{}^2}$
  - A virtuális változó *t*. eleme a bemeneti változók *t*. elemeinek négyzetösszegének a gyöke, azaz a bemeneti változók *t* időpont beli értékeiből előállított vektor hossza. A virtuális változó ezekből a hosszakból előállított értéksor. Ez az operátor olyan esetekben lehet hasznos, amikor az idősor változói valamilyen térbeli információt írnak le (pl. idegpályák helyzetét).

- VVOAngle(cos | angle, $\vec{a}$, $\vec{V}$):
  - cos mód: $\Theta_{vv} = \frac{\sum_{i=1}^{V} \Theta_{\vec{V}[i]} \vec{a}[i]}{\sqrt{\sum_{i=1}^{V} \Theta_{\vec{V}[i]}{}^2} \sqrt{\sum_{i=1}^{V} \vec{a}[i]^2}}$
  - cos mód: $\Theta_{vv} = \cos^{-1}\left(\frac{\sum_{i=1}^{V} \Theta_{\vec{V}[i]} \vec{a}[i]}{\sqrt{\sum_{i=1}^{V} \Theta_{\vec{V}[i]}{}^2} \sqrt{\sum_{i=1}^{V} \vec{a}[i]^2}}\right)$
  - Az operátor egyik bemenete egy *V* hosszú $\vec{a}$ vektor. A virtuális változó *t*. eleme a bemeneti változók *t*. elemeiből alkotott vektor és az $\vec{a}$ vektor által bezárt szög cosinusa, vagy a bezárt szög (paramétertől függően). A virtuális változó ezekből az értékekből előállított értéksor. Az előző operátorhoz hasonlóan akkor hasznos, ha térbeli információval dolgozunk.
- VVODeriv(v_i):
  - $\Theta_{vv} = \begin{cases} \Theta_{vv}[1] = \Theta_{v_i}[1] \\ \Theta_{vv}[t] = \Theta_{v_i}[t] - \Theta_{v_i}[t-1], \ t \neq 1 \end{cases}$
  - Egyváltozós VVO, a virtuális változó a bemeneti változó értéksorának a deriváltja, azaz az egyenletes mintavételezés miatt az egymást követő értékek különbsége.
- VVOInteg(v_i):
  - $\Theta_{vv} = \begin{cases} \Theta_{vv}[1] = \Theta_{v_i}[1] \\ \Theta_{vv}[t] = \sum_{k=2}^{t} \frac{\Theta_{v_i}[k] + \Theta_{v_i}[k-1]}{2}, \ t \neq 1 \end{cases}$
  - Egyváltozós VVO, a virtuális változó a bemeneti változó értéksora alatt lévő terület (integrálja), azaz az egyenletes mintavételezés miatt az egymást követő értékek átlagainak összege.

4. Fejlesztések az alap algoritmuson

Ebben a fejezetben bemutatom azokat a fejlesztéseket, amiket a ShiftTree algoritmuson végeztem, annak érdekében, hogy hatékonyabbá tegyem azt. A fejezet első részében leírom, hogy hogyan sikerült jelentősen csökkentenem a futási időt, anélkül, hogy a pontosságából bármit is veszített volna az algoritmus.

A második részben ismertetem azokat a tanítási módokat, heurisztikákat, amikkel úgy tehető pontosabbá a modell, hogy közben nem növekszik meg jelentősen a futási idő, ellentétben a korábban ismertetett MM módszerrel.

A fejezet harmadik és egyben utolsó része arról szól, hogy milyen eredményeket lehet elérni különböző nyesési módszerek alkalmazásával.

4.1. Futási idő csökkentése

A futási idő csökkentése minden algoritmus fontos, mivel így adott idő alatt több problémát oldhatunk meg, illetve adatbányászati algoritmusoknak több változatát is kipróbálhatjuk. Elméletileg a modell alapú tanulóalgoritmusoknál a tanítás idejére nincsenek szigorú korlátok, mivel általában a problémáknál a címkézés gyorsasága kritikus.

A gyakorlatban viszont könnyen előfordulhat, hogy a tanítás ideje összemérhető az adatok változásának idejével. Gyakorlati problémáknál gyakran előfordul az a jelenség, hogy az egyes osztályok jellemzői idővel változnak. Ha ez a változás gyorsabban zajlik le, mint a modellünk tanítása, akkor a felépített modell használhatatlan lesz, mivel az osztályok "régi" jellemzőit tanulta meg, így az új adatokon nem tud pontos címkézést végezni.

A 2.3.1. pontban már ismertettem, hogy egy ShiftTree csomópontban $O\left(N_{ESO}N_{CBO}{N_V}^2(N_{TR}T + N_{TR}log_2 N_{TR})\right)$ lépésben ki lehet választani a dinamikus attribútum generálásához szükséges operátorokat és változókat. Azt is megmutattam, hogy a skálázhatóság szempontjából az $O(N_{TR}log_2 N_{TR})$ tag a jelentős, mivel a többi érték egy adott problémánál tipikusan konstans.

A gyakorlatban viszont a fentiekhez hozzáadódik még annak a műveleti igénye, amikor a *CV* szerint rendezett, minden egymást követő két idősorra meghatározzuk a jósági függvény értékét, azaz a gyermek csomópontok entrópiáinak összegét. Ezt az eredeti algoritmusban $N_{ESO}N_{CBO}{N_V}^2(N_{TR} - 1)$-szer tesszük meg[7], ami a gyakorlatban kifejezetten nagy szám is lehet.

Tipikusan 100-150 ESO-t és 40-60 CBO-t használok egy probléma megoldásához, ami már egyváltozós, 100 tanítópéldát tartalmazó adatsor esetében is azt jelenti, hogy az entrópiát 400000-900000-szer kell kiszámolni. Amennyiben ezt a számot lehetne csökkenteni, jelentősen lehetne növelni az egyes csomópontok kifejtésének, és így az egész algoritmusnak, a sebességét.

Az algoritmusnak az entrópia számításon kívül nincs olyan része, ahol jelentősen lehetne csökkenteni a számításigényt, anélkül, hogy ne használnánk közelítő módszereket, és így ne vesztenénk (nagy valószínűséggel) a pontosságból.

[7] Magának az entrópia összegnek a kiszámításához nem kell újra végigmenni az összes tanítómintán, mivel az egyes gyermekekbe került osztályok számosságát inkrementálisan nyilvántarthatjuk, így az összes entrópia összeg számításának műveletigénye nem $O\left(N_{ESO}N_{CBO}{N_V}^2{N_{TR}}^2\right)$, hanem $O\left(N_{ESO}N_{CBO}{N_V}^2N_{TR}N_C\right)$, és N_C értéke alacsony.

4.1.1. Vágási helyek vizsgálatának optimalizálása

Az előzőek alapján az a feladat, hogy csak azokat a vágási helyeket vizsgáljuk meg, ahol várható, hogy a gyermek csomópontok entrópiáinak összege minimális lehet.

Állítás: Egy adott rendezés mellett a gyermek csomópontok entrópiáinak az összege csak két olyan szomszédos idősor közötti vágásnál lehet minimális, melyek különböző osztályokba tartoznak, azaz kihagyhatjuk azon szomszédos idősorok közötti vágási helyek vizsgálatát, melyek azonos osztályba tartoznak.

Bizonyítás: Vegyük egy általános, N_C osztályos osztályozási problémát, amin lefuttatjuk a ShiftTree algoritmust, és eljutunk odáig, hogy egy *CV* szerint rendeztük az idősorokat. A 4. ábra egy ilyen rendezésre mutat példát. Az egyes vékony téglalapok egy-egy idősort jelölnek, a színűk azt jelzi, hogy melyik osztályhoz tartoznak, a sorrendjük egy adott dinamikus attribútum értéke szerinti növekvő sorrend. A bizonyítás célja annak a megmutatása, hogy bármely homogén intervallumon belül vágva a 2.3. fejezetben bemutatott entrópia összeg függvény értéke nagyobb, mint az intervallumok széleinél felvett értékek minimuma.

4. ábra: Példa CV szerinti rendezésre

Vegyünk egy tetszőleges homogén intervallumot és használjuk a következő egyszerűsített jelöléseket:

- N_L: Az intervallum kezdetétől balra az idősorok száma. Azaz azon idősoroknak a száma, amik attól függetlenül, hogy az intervallum szélénél, vagy közepénél vágunk, a bal oldali gyermekbe kerülnek. Mivel minden csomópontba kerül legalább egy idősor $N_L > 0$.
- N_L^i: Az intervallum kezdetétől balra az l_i osztályba tartozó idősorok száma. Nyilvánvaló, hogy $0 \leq N_L^i \leq N_L$ és $N_L = \sum_{i=1}^{N_C} N_L^i$.
- N_L^S: Az intervallum kezdetétől balra azon idősorok száma, amelyek ugyanabba az osztályba tartoznak, mint az intervallum idősorai. $0 \leq N_L^S < N_L$, egyenlőséget nem engedünk meg, mert akkor az intervallumtól balra is csupa ugyanolyan osztályú idősor állna, azaz nem az intervallum bal szélét vettük volna.
- N_R: Az intervallum kezdetétől jobbra az idősorok száma. Mivel minden csomópontba kerül legalább egy idősor $N_R > N_I$.
- N_R^i: Az intervallum kezdetétől jobbra az l_i osztályba tartozó idősorok száma. Nyilvánvaló, hogy $0 \leq N_R^i \leq N_R$ és $N_R = \sum_{i=1}^{N_C} N_R^i$.
- N_R^S: Az intervallum kezdetétől jobbra azon idősorok száma, amelyek ugyanabba az osztályba tartoznak, mint az intervallum idősorai. $N_I \leq N_R^S < N_R$, egyenlőséget nem engedünk meg, mert akkor az intervallumtól jobbra is csupa ugyanolyan osztályú idősor állna, azaz nem az intervallum jobb szélét vettük volna.
- N_I: Az intervallum hossza. $2 \leq N_I < N$, mert nem foglalkozunk az egy hosszú intervallumoknál, mivel ott nem létezik közbenső vágás.
- X: Azon idősorok száma, amelyek az intervallumból az adott vágásnál a bal gyermekbe kerülnek. $0 \leq X \leq N_I$.
- N_L': Egy adott vágásnál a bal gyermekbe kerülő idősorok száma.
- N_R': Egy adott vágásnál a jobb gyermekbe kerülő idősorok száma.
- N: A csomópont összes idősorának a száma. $N = N_L + N_R + N_I = N_L' + N_R'$

A 2.3.1.-ben ismertetett entrópia összeg kifejezés a bevezetett jelölésekkel az alábbi formában írható fel:

$$-\sum_{X\in[L,R]}\frac{N'_X}{N}\sum_{i=1}^{N_C}\left(\frac{N_X^{i'}}{N'_X}log_2\frac{N_X^{i'}}{N'_X}\right)=$$

$$=-\frac{1}{N}\left(\sum_{\substack{i=1\\i\neq S}}^{N_C}\left(N_L^i log_2\frac{N_L^i}{N_L+X}+N_R^i log_2\frac{N_R^i}{N_R-X}\right)+(N_L^S+X)log_2\frac{N_L^S+X}{N_L+X}+(N_R^S-X)log_2\frac{N_R^S-X}{N_R-X}\right)$$

A bizonyítás során azt kell megmutatni, hogy ez a kifejezés $X=0$ vagy $X=N_I$-nél veszi fel a minimumát. első lépésben megmutatom, hogy ez igaz a nem elfajuló esetekben, azaz amikor $N_L^S\neq 0$ és $N_R^S\neq N_I$.

Alakítsuk tovább a fenti kifejezést. A szorzó léte nem befolyásolja a problémát, ezért azt elhagyom, ezután a logaritmus tulajdonságai alapján szétbontom a kifejezést.

$$-\sum_{\substack{i=1\\i\neq S}}^{N_C}\left(N_L^i log_2 N_L^i+N_R^i log_2 N_R^i\right)-(N_L^S+X)log_2(N_L^S+X)-(N_R^S-X)log_2(N_R^S-X)+$$

$$+\sum_{\substack{i=1\\i\neq S}}^{N_C}\left(N_L^i log_2(N_L+X)+N_R^i log_2(N_R-X)\right)+(N_L^S+X)log_2(N_L+X)+(N_R^S-X)log_2(N_R-X)$$

Az első szumma *X*-től független konstans, ezért a továbbiakban elhagyható. További tagok összevonása után a következő kifejezést kapjuk:

$$-(N_L^S+X)log_2(N_L^S+X)-(N_R^S-X)log_2(N_R^S-X)+(N_L+X)log_2(N_L+X)+(N_R-X)log_2(N_R-X)$$

Vegyük az első deriváltat, a logaritmus alapja miatt bejövő (pozitív) szorzót egyből elhagyva:

$$-log_2(N_L^S+X)+log_2(N_R^S-X)+log_2(N_L+X)-log_2(N_R-X)=log_2\frac{N_L+X}{N_L^S+X}+log_2\frac{N_R^S-X}{N_R-X}$$

Vegyük a második deriváltat, a szorzót itt is elhagyva:

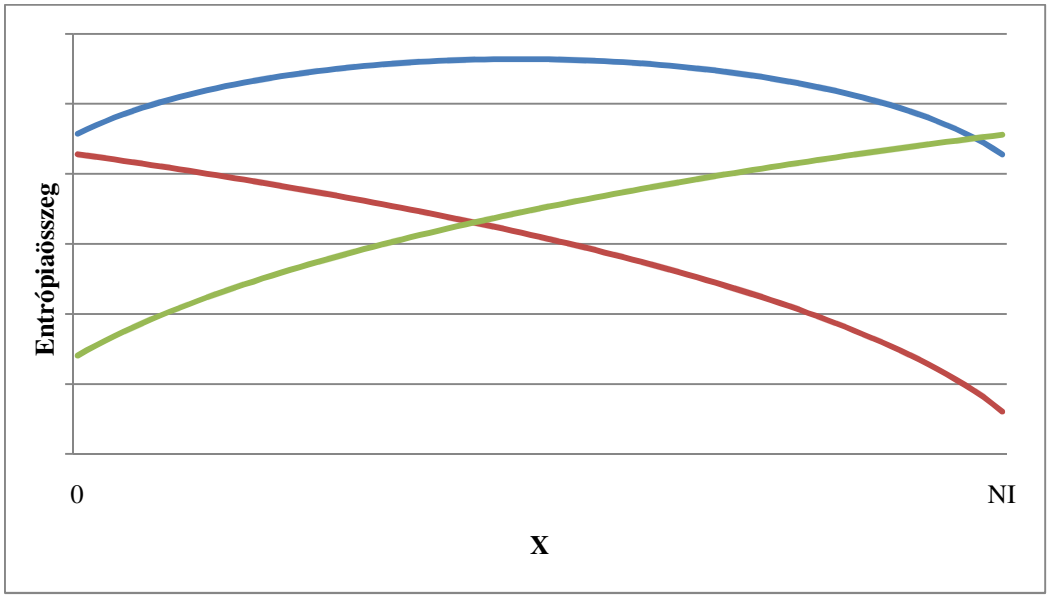
$$\frac{N_L^S+X}{N_L+X}\frac{N_L^S-N_L}{(N_L^S+X)^2}+\frac{N_R-X}{N_R^S-X}\frac{N_R^S-N_R}{(N_R-X)^2}=\frac{N_L^S-N_L}{(N_L+X)(N_L^S+X)}+\frac{N_R^S-N_R}{(N_R-X)(N_R^S-X)}<0$$

Vegyük észre, hogy mindkét tag számlálója negatív értékű, a nevezők pedig pozitívak, és a nem elfajuló esetekben egyik sem nulla *X* értékétől függetlenül. Azaz a második derivált a teljes intervallumon negatív, így ha a függvénynek van is lokális szélsőértéke, az csak lokális maximum lehet. Ezzel bebizonyítottam, hogy a nem elfajuló esetekben az entrópia összeg a homogén intervallumok valamely szélénél veszi fel a minimumát. Az entrópia összeg lehetséges függvényalakjait az intervallumon belül az 5. ábra mutatja: az intervallumon az érték vagy monoton növekvő (zöld), vagy monoton csökkenő (vörös), vagy egy lokális maximummal rendelkező görbe. Mindhárom esetben - az intervallumon - a minimumot az egyik végpontban éri el az entrópia összeg függvény.

Az elfajuló eseteknél akkor ütközünk problémába, ha az intervallum széleinél vizsgálódunk, mivel $N_L^S=0$ és $X=0$ esetén a második derivált első tagjában szerepel 0-val való osztás, $N_R^S=N_I$ és $X=N_I$ esetében pedig a másodiknál. Ha az intervallum belsejében vizsgálódunk, ott ugyanúgy igaz, hogy a második derivált negatív, mint a nem elfajuló eseteknél. Tehát a függvény minimális értéke vagy az intervallum szélén vagy attól egy lépéssel beljebb van.

Az elfajuló eseteknél vegyük a második derivált határértékét ($X\to 0$-nál, illetve $X\to N_I$-nél), ami mindkét esetben $-\infty$. Az első derivált határértékei a két esetben rendre ∞ és $-\infty$. Azaz a

függvény grafikonjának érintője mindkét esetben egy függőleges egyenes. A fentiekből következik, hogy az elfajuló esetekben is az intervallum egyik szélénél lesz a minimális érték.



5. ábra: Entrópia összeg lehetséges fügvényalakjai homogén intervallumon belül

Kimaradt az eddigi vizsgálatokból az, hogy mi van, ha a vizsgált intervallum eleje (vagy vége) egybeesik a rendezés elejével (végével). Mivel az entrópia összeg függvény szimmetrikus, elég az egyik esetet vizsgálni. Vizsgáljuk meg az, amikor az intervallum vége egybeesik a rendezés végével (így nem kell változtatni az eddigi jelöléseken). Ebben az esetben a feltételek egy kicsit megváltoznak:

- $0 \leq X < N_I$
- $2 \leq N_I \leq N$

Ez az eset nem sokban különbözik az előzőektől, csak egy kicsivel egyszerűbb: a jobb oldali gyermek entrópiája nulla lesz, mivel teljesen homogén a bele kerülő idősor halmaz. A levezetés teljesen analóg az általános esettel. Csak a baloldali gyermek entrópiájával kell foglalkozni, aminek az X-től függő része:

$$-(N_L^S + X)log_2(N_L^S + X) + (N_L + X)log_2(N_L + X)$$

A második derivált:

$$\frac{N_L^S - N_L}{(N_L + X)(N_L^S + X)} < 0$$

Elfajuló eset az $N_L^S = 0$, $X = 0$, de a második derivált ott is $-\infty$-hez tart, ha X tart a nullához, tehát a függvény minimumhelye az intervallum szélén (elején) van.

Következmény: Az entrópia összeg függvény értékét elegendő (egy adott rendezés mellett) azokon a helyeken meghatározni, ahol két különböző osztályba tartozó idősor áll egymás mellett. Az entrópia összeg kiszámításainak a száma legrosszabb esetben így is $N_{ESO}N_{CBO}{N_V}^2(N_{TR} - 1)$ marad, de a gyakorlatban jelentősen csökken a függvény értékének a kiszámításainak a száma. Különösen igaz ez, ha a problémát jól leíró operátorokat használunk, mivel minél jobban írja le az operátor a problémát, annál nagyobbak az intervallumok a rendezés után, így annál kevesebbszer kell kiszámolni az entrópia összeg függvény értékét. Ezzel a futási idő jelentősen csökkenthető és a skálázhatóság is javul. Az eredményeket a 4.1.3. pontban ismertetem.

4.1.2. Kifejtés leállítása optimumnál

Az algoritmus futási idejét tovább lehetne csökkenteni, ha az operátorpárok kipróbálását csak addig folytatnánk, ameddig meg nem találjuk a lehető legjobb vágást / dinamikus attribútumot. Sajnos azt előre nem tudhatjuk, hogy az aktuális vágás a lehető legjobb-e az adott csomópontban az adott operátorkészlet mellett. Azt viszont meg tudjuk határozni, hogy mi az entrópia összeg függvény minimális értéke a csomópontban. Ha valamely vágásnál elérjük ezt az értéket, akkor biztosak lehetünk benne, hogy jobb vágást nem találunk, maximum egy ugyanilyen jót, így a többi dinamikus attribútumot nem kell megvizsgálni. Amennyiben nem az utolsónak megvizsgált dinamikus attribútumnál értük el ezt a minimális értéket, akkor elég sok számítást megspórolhatunk.

A módszernek két szépséghibája van: az első, hogy nagymértékben épít arra, hogy mindig az első optimális vágást használjuk fel a csomópontban. Éppen ezért MM módban (és a 4.2. fejezetben bemutatott új tanítási módokban) nem használható. A másik az, hogy a legtöbb csomópontban nincs olyan vágás, amihez az entrópia összeg elérné a függvénynek a csomópontban lehetséges minimumát. De ha az adatsor és az operátorkészlet megfelelő, akkor lehetnek ilyen csomópontok, és így csökkenthető a futási idő.

Megtehetnénk azt is, hogy ha az adott vágásnál az entrópia összeg értéke közel van a lehetséges minimálishoz, akkor azt a vágást választjuk, és megállunk. Ezzel csak az a probléma, hogy nehéz általánosan definiálni azt, hogy mit értünk azon, hogy "közel van" a minimálishoz. A különféle heurisztikák használatával akár jelentősen ronthatunk a felépített modell pontosságán, ezért pontos egyenlőséget követelünk meg.

Állítás: Az entrópia összeg egy N_C osztályos osztályozási problémánál akkor minimális, ha minden, az l_i osztályhoz tartozó idősor ugyanazon csomópontba kerül (minden l_i-re) és a gyermek csomópontok mérete között a lehető legkisebb a különbség.

Bizonyítás (1. feltétel): Egy kétosztályos osztályozási problémánál egy adott csomópontban az entrópia összeg függvény minimális értéke 0. Ez akkor és csak akkor fordul elő, ha az adott vágás tökéletesen szeparál, azaz az egyik gyermekbe csak az egyik, a másik gyermekbe csak a másik osztályba tartozó idősorok kerülnek.

Egy többosztályos osztályozási problémánál is igaz az, hogy ahhoz, hogy elérjük az entrópia összeg függvény minimumát, az egyik szükséges feltétel az, hogy a vágás az egyes osztályokhoz tartozó idősorokat egy gyermekbe sorolja, azaz l_i összes idősora vagy kizárólag a bal vagy kizárólag a jobb gyermekbe kerüljön, és ne mindkettőbe. Ezt teljes indukcióval könnyen beláthatjuk:

- $N_C = 2$-re az állítás nyilvánvalóan teljesül.
- Tegyük fel, hogy $N_C = n$-re az állítás teljesül.
- Ekkor $N_C = n + 1$ osztályos problémánál képzeljünk el egy olyan operátort, ami az $N_C = n$ osztályos optimális sorrendezésbe képes úgy beékelni az l_{n+1} osztályhoz tartozó idősorokat, ahogy mi szeretnénk. Az, hogy a vágás két oldalán az új osztályhoz tartozó idősorok hogyan rendeződnek el, a vágási ponthoz közel vannak, vagy távol tőle, az az entrópia összeg függvény értéke szempontjából ekvivalens (ameddig nem változik, hogy a vágás adott oldalára mennyi ilyen idősor kerül). Szúrjuk be úgy az új idősorainkat, hogy egy egybefüggő intervallumot alkossanak. Ahhoz, hogy az entrópia összeg minimális legyen, a 4.1.1.-ben bemutatott bizonyítás alapján a vágásnak vagy az új idősorok által alkotott intervallum elejére kell esnie, vagy a végére. Tehát az összes új idősornak vagy a bal, vagy a jobboldali gyermek csomópontba kell majd kerülnie ahhoz, hogy minimális legyen az entrópia összeg.

Bizonyítás (2. feltétel): A másik szükséges feltétel azt mondja ki, hogy mely osztályoknak kell ugyanazon gyermekcsomópontba kerülnie. A következő jelöléseket használva:

- N_i: Az l_i címkéhez tartozó idősorok száma a csomópontban.
- N_L: A baloldali gyermekhez kerülő idősorok száma.
- N_R: A jobb gyermekhez kerülő idősorok száma.
- N: A csomópont összes idősorának száma. $N = N_L + N_B = \sum_{i=1}^{N_C} N_i$

A 4.1.1. egyszerűsített képletét használva kapjuk ezt:

$$-\sum_{i|\exists\langle\Theta_n, l_n=l_i\rangle\in TR_L} N_i log_2 \frac{N_i}{N_L} - \sum_{i|\exists\langle\Theta_n, l_n=l_i\rangle\in TR_R} N_i log_2 \frac{N_i}{N_R} =$$

$$= \sum_{i|\exists\langle\Theta_n, l_n=l_i\rangle\in TR_L} N_i log_2 N_L - N_i log_2 N_i + \sum_{i|\exists\langle\Theta_n, l_n=l_i\rangle\in TR_R} N_i log_2 N_R - N_i log_2 N_i$$

Vegyük észre, hogy a probléma szempontjából az N_L-t és N_R-t nem tartalmazó tagok elhagyhatóak és azt, hogy így a szummázás elvégzésével a következő kifejezést kapjuk:

$$N_L log_2 N_L + N_R log_2 N_R = N_L log_2 N_L + (N - N_L) log_2 (N - N_L)$$

Ez a kifejezés egy két kimenetű eloszlás entrópiájának a mínusz egyszerese lenne, csak a relatív gyakoriságok helyett a gyakoriságokat használjuk. Tudjuk, hogy egy ilyen entrópia függvénynek a maximuma $N_L = \frac{N}{2}$-nél van adott N mellett, így a mínusz egyszeresének minimuma is ennél az értéknél található. A függvény ismert tulajdonságai alapján a mínusz egyszeres függvény a $\left[0, \frac{N}{2}\right]$ intervallumon szigorúan monoton csökken, a $\left[\frac{N}{2}, N\right]$ intervallumon pedig szigorúan monoton nő. Az is ismert, hogy a függvény szimmetrikus. Így minél kisebb az $N_L - \frac{N}{2}$ érték, annál kisebb a fenti kifejezés értéke, és így az entrópia összeg.

Következmény: Ahhoz, hogy ki tudjuk számolni az adott csomópontban az entrópia összeg lehetséges legkisebb értékét, úgy kell szétosztani a gyermekek között az egyes osztályokat, hogy a két csomópont mérete között a lehető legkisebb különbség legyen. Ennek a megoldása visszavezethető a részösszeg [11] problémára, ami egy NP-nehéz probléma. Viszont az osztályok száma általában alacsony, a tipikus a 2-6 közötti címkeszám. Ennyi bemenet mellett a probléma megoldásának exponenciálisan skálázódó algoritmusa [11] viszonylag gyorsan lefut. Az említett algoritmus a következő:

- Bementet: $\{a_i \in \mathcal{N}\}_{i=1}^{N},\ t \in \mathcal{N}$
- Kimenete: $A = \sum_{i\in I} a_i \leq t$, A maximális a lehetséges A-k közül, $I \subseteq \{1 \dots N\}$
- Exponenciális listás algoritmus:
  - Az a_i-ket rendezzük
  - $L_0 = \{0\}$, $L'_0 = \{a_1\}$
  - $L_{k+1} = L_k \cup L'_k$
  - Az L_{k+1} lista értékeit növekvő sorrendbe rendezzük
  - $L'_{k+1} = L_{k+1} + a_{k+2}$, azaz a lista minden eleméhez hozzáadjuk a következő a_i értékét, ha egy elem értéke így nagyobb t-nél, azt az elemet töröljük a listából
  - L_n utolsó eleme lesz a maximális $A = \sum_{i\in I} a_i \leq t$

A gyorsítási módszer tehát úgy néz ki, hogy minden nem-levél csomópont kifejtése előtt kiszámítjuk a részösszeg probléma megoldását az $\{N_i\}_{i=1}^{N_C}$, $t = \frac{\sum_{i=1}^{N_C} N_i}{2}$ bementre, így megkapjuk az optimális szétosztásnál az egyik gyermekcsomópont méretét [8]. Ennek segítségével kiszámoljuk az entrópia összeg függvény értékét, így megkapjuk a minimális értéket. A vágások vizsgálata során, ha találunk egy olyan vágást, ahol az entrópia összeg értéke megegyezik a minimálissal, akkor abbahagyjuk a további vágások / dinamikus attribútumok vizsgálatát.

4.1.2.1. Minimum elérés után vizsgálatok korlátozása

Ahogy fentebb említettem, a 4.1.2.-ben ismertetett gyorsítási módszer egyik hátránya az, hogy több lehetséges optimális vágás közül mindig az elsőt fogja választani. Jóllehet az eredeti ShiftTree algoritmus is ezt tette, ott ez a választás opcionális volt, és láthattuk, hogy pl. az MM tanítási mód felhasználja az összes optimális vágást. A 4.2. fejezetben ismertetek még újabb tanítási módokat, amik szintén figyelembe veszik az összes optimális vágást, mert például azok közül választanak.

Ahhoz, hogy ezek a módszerek együtt tudjanak működni, módosítani kell a 4.1.2.-ben ismertetett gyorsítási módszert. Miután elérjük a csomópontban a minimumot, nem állunk le a dinamikus attribútumok vizsgálatával. Viszont jelentősen leszűkítjük a megvizsgált vágásokat, így még mindig csökkenthetjük a számítási kapacitást az eredeti megoldáshoz képest.

Tudjuk, hogy a csomópontban a már megtalált minimális entrópia összegű vágásnál semmiképpen sem lehet jobb vágás, viszont lehet vele egyenlő jóságú. Azt is tudjuk, hogy ez a vágás milyen méretű gyermek csomópontokat eredmények. Ezért megtehetjük azt, hogy miután elértük az entrópia összeg függvény minimumát a csomópontban, az elkövetkező dinamikus attribútumok alapján történő rendezéseket csak két vágási pontnál vizsgáljuk: annál a két pontnál ahol az egyik gyermek mérete megegyezik a listás algoritmus által visszaadott értékkel. Ha valamely pontban az entrópia összeg értéke megegyezik a minimálissal, akkor találtunk egy új optimális vágást a régi(ek) mellé.

4.1.3. A gyorsítási módszerek hatása

Az alfejezetben kidolgozott módosításokat az UCR és a Ford adatokon teszteltem. A model pontos operátorkészlete a VI. függelékben található "bővített operátorkészlet" név alatt. A 3. táblázat foglalja össze a mérési eredményeket. Minden módszernél két oszlop látható: az első a futási idő másodpercekben, a második pedig az eredeti algoritmushoz képest a futási idő százalékos változása.

A 4.1.1.-ben bemutatott, a vizsgált vágási helyek terét szűkítő módszerrel az alap algoritmus futási idejét közel 20%-kal lehet átlagosan csökkenteni. A legkisebb javulás a kisebb adatsoroknál (kivéve Coffee) látható, de ott is nagyobb, mint 8,5%. Közepes és nagy tanítóhalmazzal rendelkező problémákon a javítás mértéke meghaladja a 10%-ot.

Annak ellenére, hogy az eredeti tanítási módon kívül mással nem kompatibilis, a 4.1.2.-ben ismertetett, az optimumnál megálló módszert is megvizsgáltam. Az eredményekből az látszik, hogy ennek a módszernek a hatékonysága nagyon adatsor (és modell) függő. Van olyan probléma, ahol közel 96,7%-ot képes gyorsítani, de van ahol csak alig 4,18%-ot. A módszer azoknál a problémáknál nagyon hatékony, ahol a kialakított modellben olyan csomópontok

[8] Ha egy csomópontban több, mint 20 különböző címkéhez tartozó osztály van, a számítás az idő- és tárigénye miatt nem hajtom végre. Ha egy feladatban több, mint 20 féle címke van (pl. 50Words, Adiac), csak azoknak a csomópontoknak a kifejtésén gyorsítok a módszerrel, ahova már 20-nál kevesebb különböző címke kerül.

vannak, amik a tanítóhalmaz nagy részén dolgoznak, és az idősorokat tökéletesen szeparálják, azaz az entrópia összeg értéke eléri a minimumát. Ilyen modellje lesz többek között például a Trace-nek és a Wafer-nek is. A javulás mértéke átlagosan közel van a 40%-hoz.

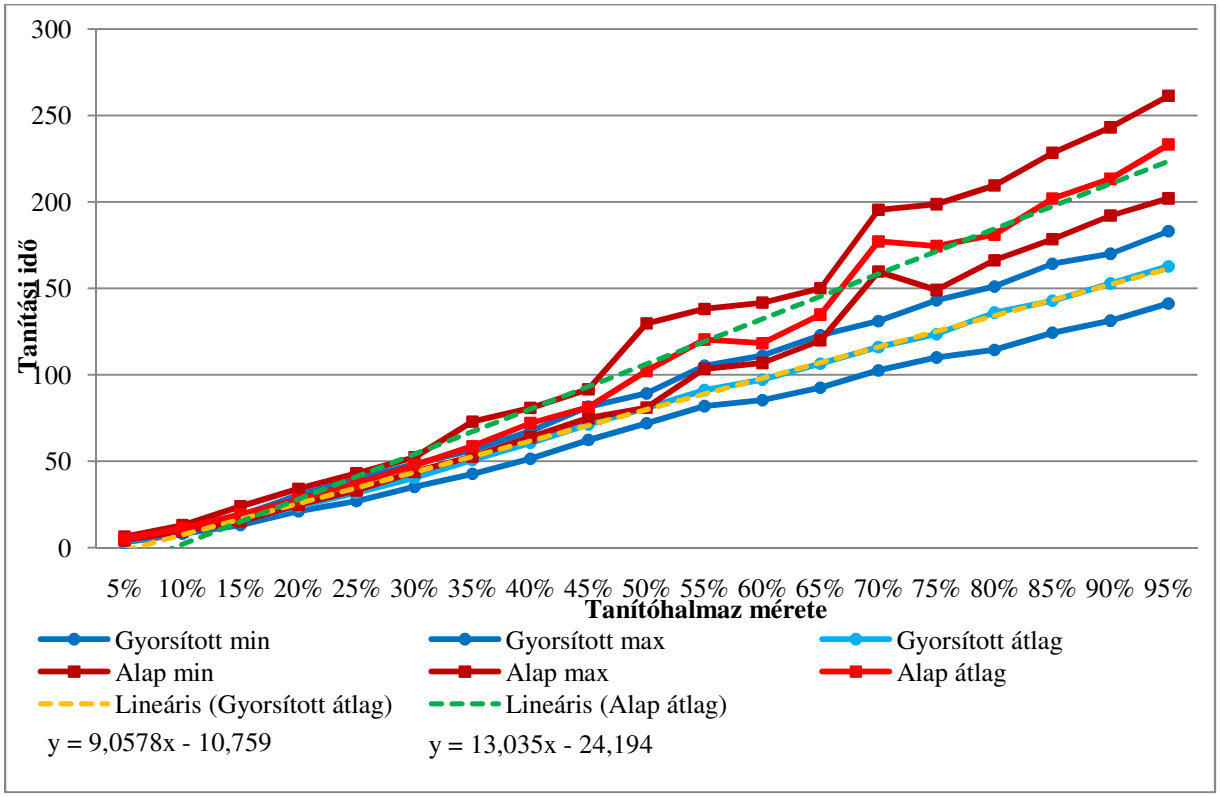
A 4.1.2.1.-ben ismertetett módszer, ami a 4.1.2.-beli módszernek a lassabb verziója, az előzővel teljesen analóg eredményeket mutat, csak a hatékonysága alacsonyabb. Ez is azokon az adatsorokon a leghatékonyabb, ahol kialakult modellben a gyökérhez közeli csomópontokban létezik tökéletes vágás. A javulás mértéke kevesebb, mint a 4.1.1. módszeré, kb. 13%.

Adatsor	Eredeti ShiftTree	Vágási helyek terének szűkítése		Kifejtés leállítása optimumnál		Számítások csökkentése optimumnál		Gyorsított algoritmus	
	Futási idő	Futási idő	Javulás	Futási idő	Javulás	Futási idő	Javulás	Futási idő	Javulás
50Words	47,4192	34,7226	-26,78%	34,6389	-26,95%	38,1653	-19,52%	33,9944	-28,31%
Adiac	29,7609	22,1863	-25,45%	23,0646	-22,50%	25,0895	-15,70%	21,8091	-26,72%
Beef	0,5741	0,5246	-8,62%	0,4734	-17,54%	0,5651	-1,57%	0,5173	-9,90%
CBF	0,2458	0,1725	-29,81%	0,0087	-96,45%	0,1528	-37,82%	0,1446	-41,18%
Coffee	0,2641	0,1838	-30,42%	0,1391	-47,31%	0,2010	-23,91%	0,1783	-32,48%
ECG200	0,3334	0,2787	-16,40%	0,0501	-84,97%	0,2588	-22,37%	0,2423	-27,31%
FaceAll	25,1091	18,4045	-26,70%	20,0357	-20,21%	21,4706	-14,49%	18,2633	-27,26%
FaceFour	0,3163	0,2350	-25,70%	0,1206	-61,87%	0,2364	-25,25%	0,2211	-30,11%
Fish	6,4310	4,5783	-28,81%	4,5799	-28,78%	5,1703	-19,60%	4,5140	-29,81%
GunPoint	0,3655	0,2704	-26,00%	0,1857	-49,20%	0,3002	-17,86%	0,2671	-26,90%
Lighting2	1,4174	1,2675	-10,58%	1,0556	-25,53%	1,3903	-1,92%	1,2607	-11,06%
Lighting7	1,6154	1,4392	-10,91%	1,3906	-13,92%	1,5844	-1,92%	1,4115	-12,62%
OliveOil	0,3268	0,2895	-11,41%	0,0953	-70,83%	0,2838	-13,15%	0,2659	-18,64%
OSULeaf	6,6165	5,8915	-10,96%	5,6802	-14,15%	6,5392	-1,17%	5,8475	-11,62%
SwedishLeaf	19,3132	16,2720	-15,75%	17,9762	-6,92%	19,0619	-1,30%	16,0933	-16,67%
SyntheticControl	4,0480	3,1804	-21,43%	2,6630	-34,21%	3,5450	-12,42%	2,9673	-26,70%
Trace	1,0166	0,8390	-17,46%	0,1554	-84,72%	0,8160	-19,73%	0,7714	-24,12%
TwoPatterns	11,0925	9,7317	-12,27%	9,1985	-17,07%	10,6010	-4,43%	9,6154	-13,32%
Wafer	4,3477	3,4586	-20,45%	0,1441	-96,69%	3,4042	-21,70%	3,3000	-24,10%
Yoga	9,2397	8,0914	-12,43%	8,8537	-4,18%	9,2180	-0,24%	8,0105	-13,30%
FordA	250,0950	202,1090	-19,19%	238,2520	-4,74%	242,0020	-3,24%	200,5160	-19,82%
FordB	214,9350	174,5090	-18,81%	195,4850	-9,05%	197,8040	-7,97%	173,5160	-19,27%
Összesítve	**634,8830**	**508,6357**	**-19,89%**	**564,2462**	**-11,13%**	**587,8597**	**-7,41%**	**503,7268**	**-20,66%**
Max javulás			***-30,42%***		***-96,69%***		***-37,82%***		***-41,18%***
Min Javulás			***-8,62%***		***-4,18%***		***-0,24%***		***-9,90%***
Átlagos javulás			***-19,38%***		***-38,08%***		***-13,06%***		***-22,33%***

3. táblázat: Gyorsítási módszerek hatása

Az alfejezetben ismertetett módszerek egymással kombinálhatóak. Gyorsított algoritmusnak nevezem azt a változatot, amikor egyszerre használom a vágási helyek terének szűkítését (4.1.1.) és a számítások csökkentését az optimum elérése után (4.1.2.1.). Látható, hogy sajnos a gyorsítások mértéke nem adódik össze, viszont a kombinált módszer még így is jobb, mint bármelyik összetevője egymagában. A minimális javulás ezeken az adatsorokon közel 10%-os, a maximális pedig meghaladja a 40%-ot. Az átlagos, 20% körüli gyorsulás kifejezetten jónak mondható.

Az abszolút futási idő mellett az is fontos, hogy az algoritmus a tanítóhalmaz méretének növekedésével hogyan skálázódik. Ahogy 2.3.1. pontban kifejtettem, hogy a teljes modell építésének az ideje nagyban függ a kialakult modell struktúrától, a tanítópontok eloszlásától az egyes csomópontokban, stb.



6. ábra: A ShiftTree skálázódása a FordA adatsoron

A 6. ábra mutatja az alap és a gyorsított ShiftTree skálázódását. A grafikonon a legnagyobb tanítóhalmazzal rendelkező, FordA adatsorhoz tartozó mérések láthatóak, de az eredmények az összes adatsor esetében nagyon hasonlóak. A kisebb méretű adatsoroknál a görbék kicsit zajosabbak a kevés tanítópont (és így a gyakran változó modellstruktúrák) miatt. A mérések keresztvalidációval készültek. A mérési pontok a tanítóhalmaz 5, 10, ..., 95 százalékának felhasználásával készültek. Egy mérés során rétegelt mintavétel segítségével (azaz az osztályok eloszlását megőrizve) véletlenszerűen kiválasztottam a tanítópontok X%-át, amin lefuttattam a tanítási algoritmust. Ezt 20-szor megismételtem és az így kapott eredmények közül kiválasztottam a minimumot, maximumot (sötétebb vonalak) és az átlagot.

Látható, hogy mind a minimumok, mind a maximumok, mind az átlagok a tanítópontok számával lineárisan skálázódnak, mind a gyorsított, mind az alap algoritmus esetében, ami kifejezettem előnyös tulajdonság. Igaz az alap algoritmusnál jóval nagyobb az egyes mérések közötti szórás és az átlagos eredmények sem illeszkednek olyan jól a lineárisra, mint a gyorsított algoritmus esetében. A zajosság abból adódik, hogy az egyes tanítóhalmazokon különböző struktúrájú modellek alakulhatnak ki, és az alap algoritmusnál ezek erősebben változó futási időt eredményeznek. A gyorsított algoritmus mérési pontjai kifejezetten jól illeszkednek a behúzott lineáris vonalra, és az összetartozó minimumok és maximumok között is viszonylag kicsi az eltérés.

A különböző adatsoroknál a lineáris meredeksége eltérő, de a gyorsított algoritmus skálázódásának meredeksége minden esetben alacsonyabb, mint az alap algoritmusé. Azaz ahogy nő a tanítóhalmaz mérete, a gyorsított algoritmussal abszolút értékben egyre több időt takaríthatunk meg.

4.2. Tanítási módok

A 2.3.-ban bemutatott tanítási algoritmus a csomópontban az optimális vágások közül minden esetben az elsőt választja. Az, hogy melyik lesz az első, az operátorok sorrendjétől függ. Ez, azon kívül, hogy nem túl kifinomult megoldás, nem is minden esetben optimális a teszthalmaz szempontjából. Viszont ezek között a vágások között a címkéket és a címkék eloszlását felhasználva nem lehet különbséget tenni, mivel ugyanolyanok.

Egy korai megoldás a problémára a 2.4.1. pontban bemutatott MM tanítás. A módszer hátránya, hogy a tanítási idő a sokszorosára nő, illetve a tanítás során felépített modell mérete is óriási lesz. Ráadásul minél több operátort használunk, a probléma annál súlyosabb, mivel annál valószínűbb, hogy több olyan dinamikus attribútum lesz, ami ekvivalensen jó sorrendezéseket eredményez. A másik probléma a modellkiválasztás problémája egyszerű nyeséssel. Ha rendelkezésünkre áll egy megfelelő validációs halmaz, vagy a tanítóhalmazból ez leválasztható, akkor megtehető a kiválasztás, de ha nincs ilyen halmaz, vagy a tanítóhalmaz kicsi (mint a dolgozatban használt adatsorok többségénél), akkor a teljes többszörös modellt meg kell tartanunk, és többségi szavazással kell címkéznünk. Ilyenkor a modell sokszor nagyobb méretű, mint maga a tanítóhalmaz, ami filozófiai problémákat is felvet, mivel nem biztos, hogy értelmes egy adatsort önmagánál nagyobb modellel leírni.

Ebben az alfejezetben bemutatom azokat a tanítási módszereket, amiknek segítségével az MM módszer pontosságát meg lehet közelíteni, anélkül, hogy a futási idő és/vagy a modell mérete a sokszorosára nőne. A fejezet csak a módszerek bemutatását tartalmazza, a numerikus eredmények a 4.4. alfejezetben találhatóak.

4.2.1. Heurisztika alkalmazása választásnál

A feladat tehát valahogy kiválasztani az optimális vágások közül egyet, csak a tanítóhalmaz felhasználásával. Definiáljunk egy mértéket, hogy mennyire tekintünk egy vágást biztosnak és válasszuk a legbiztosabb optimális vágást. A vágás biztossága alatt azt értem, hogy mennyire valószínű, hogy egy olyan idősor, amelyiknek az egyik gyermekcsomópontba kellett volna kerülnie, az valóban oda kerül, és nem a másik gyermekcsomópontba.

Természetesen ezt sem lehet előre tudni, viszont meg lehet nézni, hogy egy vágásnál mekkora a biztonsági sáv, azaz a *TV* érték és az ahhoz legközelebb lévő idősor dinamikus attribútumának értéke. Ha ez az érték nagy, akkor azt feltételezzük, hogy az adott vágás nagyobb biztonsággal sorolja jó a oldalra az idősorokat, kisebb valószínűséggel fordul elő, hogy az idősoron lévő zaj miatt a határ érték rossz oldalára kerül a kiszámított dinamikus attribútuma.

A 2. algoritmus 27. sora helyett a következő algoritmusrészlet választja ki a legjobb vágást:

27a: $\langle j'_q, v'_q, k'_q, w'_q, m'_q \rangle_{q=1}^{Q} = \{\langle j, v, k, w, m \rangle \mid E^{j,v,k,w,m} = min_{j,v,k,w,m}(E^{j,v,k,w,m})\}$

27b: $\langle j', v', k', w', m' \rangle = argmax_{j'_q, v'_q, k'_q, w'_q, m'_q} \left(H \left(\left\{ CV_n^{j'_q, v'_q, k'_q, w'_q} \right\}_{n=1}^{N_{TR}}, TV \right) \right)$

3. algoritmus: Heurisztika alkalmazása a tanításban

Azaz először kiválasztjuk az összes olyan vágást, amihez minimális entrópia összeg tartozik a csomópontban. Majd második lépésként ezek közül kiválasztjuk azt, amelyik a H heurisztikával a legnagyobb értéket adja. A 3. algoritmus 27b. sorában megjelenő a *H* heurisztika, ami azt mondja meg, hogy mennyire biztonságos a vágás, azaz mennyire különülnek el a különböző gyermekcsomópontokba kerülő idősorok a választott dinamikus attribútum alapján. Ez nem csak a biztonsági sáv abszolút értékétől függ, hanem attól is, hogy mekkora az attribútumok nagyságrendje ehhez képest. A heurisztika bemenete a vágás küszöb értéke (*TV*) és a vágás során felhasznált dinamikus attribútum értéke a csomópont összes

idősoránál. Több mértékkel is kísérleteztem, és végül az alábbi három heurisztikát találtam jól használhatónak (CV_i^R és CV_i^L jelöli a jobb, illetve a bal csomópontba kerülő idősorok dinamikus attribútumainak értékét):

$$H_1 = \frac{min_i\{CV_i^R\} - max_i\{CV_i^L\}}{max_i\{CV_i^R\} - min_i\{CV_i^L\}}$$

$$H_2 = \frac{\sum_{n=1}^{N_{TR}} |TV - CV_n|}{max_i\{CV_i^R\} - min_i\{CV_i^L\}}$$

$$H_3 = \frac{H_1}{H_2} = \frac{min_i\{CV_i^R\} - max_i\{CV_i^L\}}{\sum_{n=1}^{N_{TR}} |TV - CV_n|}$$

A H_1 heurisztika alkalmazásával működő tanítási módot SM+ -nak, a H_2 alkalmazásával működőt pedig SM++ -nak, a H_3 alapút SM3+ -nak neveztem el. Az SM+ algoritmus csomópontonkénti plusz műveletigénye konstans, az SM++ és SM3+ műveletigénye N_{TR}. Egyik módszer sem változtat a skálázódás mértékén, és a gyakorlatban egyik módszer sem növeli meg jelentősen az algoritmus futási idejét, az időbeli eltérést nem lehet kimutatni, mert az idő mérésének hibahatárán belül van.

A H_1 heurisztika a biztonsági sáv kétszeresének és a sorrendezés két vége közötti eltérésnek a hányadosa. Azaz Az attribútumok nagyságrendjének a legkisebb és a legnagyobb érték különbségét veszi. Ez a módszer akkor tévedhet, ha a sorrendezés elején és/vagy a végén kiugróan kicsi illetve nagy értékek vannak. Ebben az esetben a heurisztika alulsúlyozza az adott attribútum szerinti vágást.

A H_2 heurisztika nem csak a küszöb értékhez legközelebbi értékek távolságát vizsgálja, hanem az összes idősor dinamikus attribútumának értékére megnézi, hogy az milyen messze van a küszöbértéktől, majd ezt normálja a legtávolabbi elemek távolságával. Ez a heurisztika a küszöbértéktől kiugróan messze lévő pontok esetén felülbecsüli a vágás biztonságosságát.

Ha nincsenek outlier pontok, akkor mindkét heurisztika közel helyes képet ad a vágás biztonságosságáról. A két heurisztika kombinálásának (átlagolás, harmonikus közép, stb.) nem volt különösebb hatása, vagy az egyik, vagy a másik heurisztikára jellemző eredmények születtek.

A H_3 heurisztika a biztonsági sáv kétszeresének és az összes pontnak a biztonsági sávtól való távolságösszegének a hányadosa. A három közül ez a mérték a legkevésbé érzékeny a túl nagy/kicsi értékekre.

4.2.2. Heurisztika és korlátozott többszörös modellezés

A gyakorlatban az MM tanítási módot korlátozni kell. Többféle korlátozási lehetőség is létezik, az egyik ezek közül az, hogy megszabjuk, hogy összesen hány modellt tartalmazhat egy MM ShiftTree. Ha elérjük ezt a korlátot a tanítás során, akkor az SM tanítás szerint fejtjük ki a további kifejtendő csomópontokat. A fa magasabb szintjein lévő csomópontok fontosabbak, mint a lejjebb lévők, egyrészt mivel nagyobb hatással vannak a címkézésre, másrészt mivel még sok tanítópont alapján lettek kifejtve, azt reméljük, hogy az adatsorra vonatkozó általános tulajdonságokat írják le. Ennek az eredménye az is, hogy a magasabb szinten lévő csomópontokban kisebb valószínűséggel lesznek azonosan jó vágások, de ha mégis, akkor azok mind fontosak lehetnek. A csomópontok szélességi sorrend szerinti kifejtése azért fontos, mert ha egy MM tanítási folyamatban korlátozzuk a konkurens modellek számát, akkor ilyen kifejtés mellett a magasabb szinten lévő csomópontokban tartjuk meg az elágazásokat.

A másik korlátozási lehetőség az, hogy megszabhatjuk, hogy egy csomópontból maximum mennyi párhuzamos részfa indulhat, azaz mennyi azonosan jó vágást engedünk meg maximum. Természetesen a két feltétel egyszerre is használható. Az utóbbi feltételnél belefuthatunk a sima SM tanításnál is előjövő problémába, hogy ha több optimális vágást találunk, mint amennyit megtarthatunk, akkor melyeket tartsuk meg. Az előző pontban tárgyalt heurisztikák használata itt is jó eredményeket hozhat.

4.3. Nyesési módszerek

A 2.4.2. pontban már esett szó egy egyszerű utónyesési eljárásról, ami jó validáló halmaz mellett javíthat a modell pontosságán, csakhogy ilyen validáló halmazt nehéz találni, főleg, ha eleve kevés tanítópont áll a rendelkezésünkre. Így a módszer leginkább ellenőrzésre volt jó, azáltal, hogy megmondta, hogy az aktuális modellnek mely részfája illeszkedik a legjobban a teszthalmazra, és mekkora az illeszkedés mértéke. Ebben a fejezetben két olyan, döntési fáknál elterjedt utónyesési módszert mutatok be, amit a ShiftTree algoritmusba is beépítettem.

4.3.1. Szignifikancia alapú nyesés

A szignifikancia vagy $\chi 2$ [12] alapú nyesés egyike a legrégebben használt döntési fa nyeséseknek. Ez a módszer egy statisztikai próbával dönti el, hogy a vágásunk az adott csomópontban jelentős vagy jelentéktelen. Amennyiben a vágást jelentéktelennek ítéli, a csomópontból kiinduló részfát lenyesi. Ebből következően nem csak utó-, hanem előnyesésként is alkalmazható, A hipotézisünk az, hogy a vágásunk nem szignifikáns. Bináris döntési fa esetén N_C lehetséges osztály esetén az alábbi értéket számolja ki egy csomópontra:

$$D = \sum_{i=1}^{N_C} \frac{(\hat{p}_L - \mathrm{PLabels}_L(l_i))^2}{\hat{p}_L} + \frac{(\hat{p}_R - \mathrm{PLabels}_R(l_i))^2}{\hat{p}_R}$$

$$\hat{p}_X = \frac{\sum_{i=1}^{N_C} \mathrm{PLabels}_X(l_i)}{\sum_{i=1}^{N_C} \mathrm{PLabels}(l_i)} \mathrm{PLabels}(l_i)$$

Az L és R indexek azt jelölik, hogy az adott változó a megfelelő gyermekcsomóponthoz tartozik. A D érték lényegében azt fejezi ki, hogy az egyes osztályok aránya a szülő és a gyermek csomópontokban mennyire tér el egymástól. Ha nem tér el jelentősen, akkor a vágás nem jelentős. A fent leírt képlet kiterjeszthető többosztályos esetre nem bináris döntési fára is.

Bináris döntési fa esetén D értéke 1 szabadságfokú $\chi 2$ eloszlást követ, tehát egy $\chi 2$ próbát hajtunk végre, ami alapján eldöntjük, hogy helyes volt-e a feltételezésünk, azaz tényleg jelentéktelen-e a vágás. Mint a statisztikai próbáknál általában, itt is szükségünk van egy szignifikancia értékre. Minél kisebb a szignifikancia érték, annál szigorúbb a nyesés, annál erősebb a feltétel, amit egy vágásnak teljesítenie kell, hogy megtartsuk.

4.3.2. Komplexitás-hibaarány alapú nyesés

A komplexitás-hibaarány alapú nyesés [13] arra optimalizál, hogy a csomópontból induló részfa lenyesésével a tanítóhalmazon elkövetett hiba és a modell komplexitása között egy optimális trade-off-ot találjon. Megvizsgálja, hogy az adott csomópont bezárásával (a hozzá tartozó részfa levágásával), illetve kifejtésével mekkora a komplexitás és a hibaarány.

Jelöljük T_n -nel a vizsgált csomópontból induló részfát, és $\{T_n\}$ -nel magát a vizsgált csomópontot. A modell komplexitását közelítsük a csomópontból kiinduló részfa leveleinek számával. Nyilvánvaló, hogy $|\{T_n\}| = 1$. A hibaarányt a rosszul osztályozott tanítópontok és a

teljes tanítóminta arányával közelítjük, azaz $R(\{T_n\}) = \left(1 - max_i\left\{PLabels_{\{T_n\}}(l_i)\right\}\right)\frac{N_{TR\{T_n\}}}{N_{TR}}$.
A részfa hibaarányát hasonlóan számolhatjuk a leveleikre vett hibaarányok összegeként. Vezessünk be egy α szorzót, ami azt mutatja, hogy az általunk definiált veszteségfüggvényben a komplexitás milyen mértékben skálázódik a hibaarányhoz képest. Így megkaphatjuk a csomópont és a belőle induló részfa költségét. Adott α érték mellett mindkettő kiszámolható, és ha a csomópont költsége nagyobb, mint a részfáé, akkor a részfát nem szabad lenyesnünk. Ha a részfa költsége nagyobb, mint a csomóponté, akkor viszont a csomópontot be kell zárnunk (a részfát le kell nyesnünk).

Egy jó α érték meghatározása nehéz, ráadásul probléma függő, ezért inkább azt tesszük, hogy megvizsgáljuk, hogy egy adott csomópontnál milyen α értékre lenne egyenlő a csomópont és a belőle induló részfa költsége, ezt az értéket α_n-nel jelöljük:

$$R(T_n) + \alpha_n|T_n| = R(\{T_n\}) + \alpha|\{T_n\}| = R(\{T_n\}) + \alpha$$

$$\alpha_n = \frac{R(\{T_n\}) - R(T_n)}{|T_n| - 1}$$

Ha valaki megmondja nekünk az α paraméter értékét, akkor az összes olyan csomópontot, ahol $\alpha_n < \alpha$, be kell zárni, és az összes olyat, ahol $\alpha_n > \alpha$, nem szabad bezárni. Ebből már következik, hogy először mindig a legkisebb α_n értékkel rendelkező csomópontokat érdemes bezárnunk. Az algoritmus váza tehát a következő módon épül fel:

- Számoljuk ki α_n értékét az összes nem levél csomópontra
- Vegyük a legkisebb α_n értékkel rendelkező csomópontot és nyessük le a belőle induló részfát
- Ha nem értük el a leállási feltételt, akkor kezdjük az első ponttól

Mivel egy csomópont becsukásával annak szüleihez tartozó részfáknak mind a komplexitása, mind a hibaaránya megváltozik, ezért minden nyesés után újra kell számolni az α_n értékeket. Az algoritmus képlékeny pontja a leállási feltétel. Ha van egy jó validációs halmazunk, akkor megtehetjük, hogy minden egyes nyesés után megvizsgáljuk a modell hibaarányát a validációs halmazon, és amennyiben ez egy küszöb alá esik, akkor leállunk, vagy ha ez már nem csökken tovább, akkor leállunk.

Jó validációs halmaz viszont nem áll rendelkezésemre a legtöbb adatsor esetén. Ezért az algoritmusnak egy olyan változatát is implementáltam, ahol csak akkor állunk le, ha a gyökér csomópont α_n értéke a legkisebb (a gyökeret értelemszerűen nincs értelme bezárni, mert úgy a teljes modell elveszik). Ez a változat nem használható hatékonyan nyesésére, mivel várhatóan túlnyesi a fát. De megvizsgálható, hogy milyen α határ értékek mellett hogyan változik a modell pontossága.

4.4. Numerikus eredmények

Ebben az alfejezetben ismertetem a különböző tanítási módok és a nyesés segítségével elérhető eredményeket az UCR, Ford és AE adatsorokon. A kísérletek többségéhez a VI. függelékben leírt bővített operátorkészletet használom. Összehasonlítási alapként az 1-NN szomszéd módszer eredményeit is megmutatom, valamint a fejlesztetlen ShiftTree eredményeit is összehasonlítom a fejlesztett változattal. Az előbbi változat a VI. függelékben leírt alap operátorkészletet használja. A fejezet végén kitérek arra, hogy az algoritmus mennyire megbízható, azaz a modellek becsült pontosság értékei milyen viszonyban vannak az új adatokon elért osztályozási pontossággal.

4.4.1. Tanítási módok összehasonlítása

Adatsor	1-NN Euklideszi		1-NN DTW		Régi operátorok		SM		SM+		SM++		SM3+	
	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság
50Words	287	63,08%	308	67,69%	161	35,38%	232	50,99%	244	53,63%	232	50,99%	228	50,11%
Adiac	239	61,13%	224	57,29%	191	48,85%	218	55,75%	220	56,27%	200	51,15%	206	52,69%
Beef	16	53,33%	16	53,33%	16	53,33%	15	50,00%	17	56,67%	12	40,00%	12	40,00%
CBF	767	85,22%	880	97,78%	803	89,22%	841	93,44%	848	94,22%	873	97,00%	876	97,33%
Coffee	21	75,00%	20	71,43%	19	67,86%	26	92,86%	23	82,14%	23	82,14%	23	82,14%
ECG200	88	88,00%	82	82,00%	78	78,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%
FaceAll	1206	71,36%	1411	83,49%	1038	61,42%	1103	65,27%	1108	65,56%	1093	64,67%	1084	64,14%
FaceFour	69	78,41%	76	86,36%	42	47,73%	57	64,77%	62	70,45%	65	73,86%	60	68,18%
Fish	137	78,29%	135	77,14%	116	66,29%	136	77,71%	130	74,29%	129	73,71%	133	76,00%
GunPoint	137	91,33%	126	84,00%	117	78,00%	146	97,33%	143	95,33%	147	98,00%	147	98,00%
Lighting2	46	75,41%	48	78,69%	47	77,05%	40	65,57%	44	72,13%	41	67,21%	41	67,21%
Lighting7	42	57,53%	40	54,79%	46	63,01%	44	60,27%	46	63,01%	47	64,38%	44	60,27%
OliveOil	26	86,67%	26	86,67%	24	80,00%	22	73,33%	20	66,67%	22	73,33%	22	73,33%
OSULeaf	125	51,65%	128	52,89%	122	50,41%	143	59,09%	136	56,20%	130	53,72%	130	53,72%
SwedishLeaf	493	78,88%	476	76,16%	392	62,72%	476	76,16%	480	76,80%	495	79,20%	483	77,28%
SyntheticControl	264	88,00%	290	96,67%	276	92,00%	280	93,33%	276	92,00%	276	92,00%	276	92,00%
Trace	76	76,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%
TwoPatterns	3627	90,68%	4000	100,00%	3758	93,95%	3987	99,68%	3994	99,85%	3998	99,95%	3985	99,63%
Wafer	6136	99,55%	6032	97,86%	6022	97,70%	6156	99,87%	6163	99,98%	6164	100,00%	6164	100,00%
Yoga	2491	83,03%	2478	82,60%	1996	66,53%	2461	82,03%	2559	85,30%	2560	85,33%	2567	85,57%
FordA	901	68,26%	941	71,29%	1055	79,92%	1233	93,41%	1229	93,11%	1230	93,18%	1231	93,26%
FordB	482	59,51%	531	65,56%	545	67,28%	537	66,30%	543	67,04%	548	67,65%	555	68,52%
AE	N/A	N/A	N/A	N/A	300	81,08%	305	82,43%	287	77,57%	302	81,62%	309	83,51%
UCR összes	**16293**	**87,59%**	**16896**	**90,83%**	**15364**	**82,59%**	**16583**	**89,15%**	**16713**	**89,85%**	**16707**	**89,81%**	**16681**	**89,67%**
Ford összes	**1383**	**64,93%**	**1472**	**69,11%**	**1600**	**75,12%**	**1770**	**83,10%**	**1772**	**83,19%**	**1778**	**83,47%**	**1786**	**83,85%**
Összes	**17676**	**83,76%**	**18368**	**87,04%**	**17264**	**81,81%**	**18658**	**88,42%**	**18772**	**88,96%**	**18787**	**89,03%**	**18776**	**88,98%**

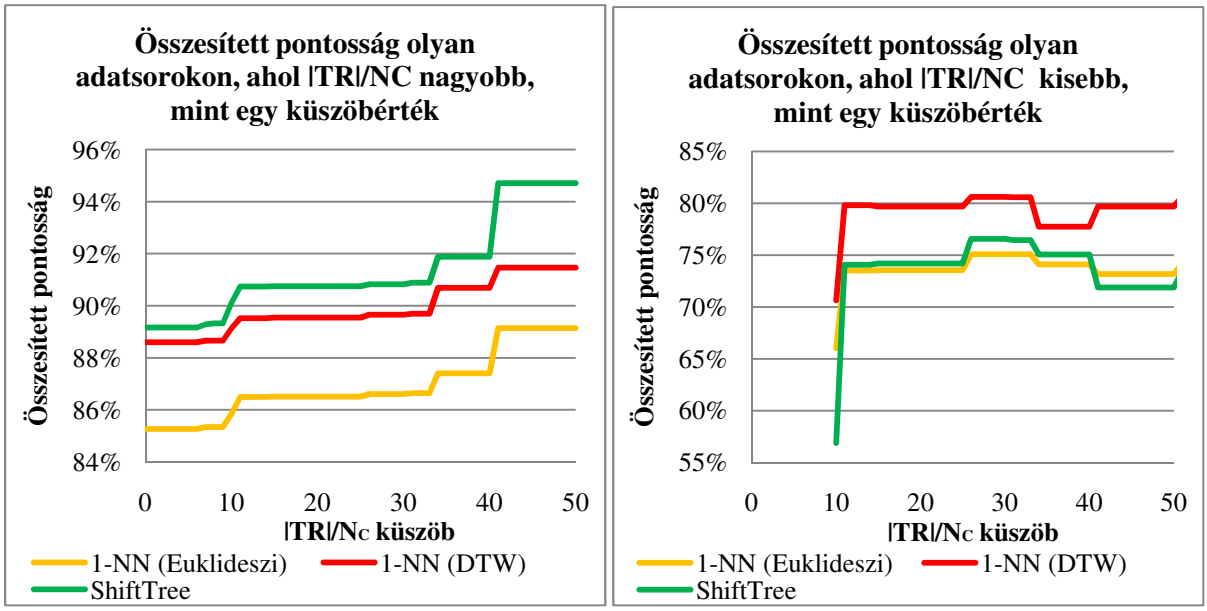
4. táblázat: Egyszerű tanítási módok eredményei

A 4. táblázat mutatja a különböző (egyszerű) tanítási módok által elért eredményeket. Az idősorok eltérő hossza és a több változó miatt az alap 1-NN algoritmus az AE idősort nem tudta osztályozni. Az eredményekből jól látszik, hogy az új operátorok bevezetésével jelentős pontosságnövekedés érhető el. Érdekes megfigyelni, hogy így is van néhány olyan adatsor, ahol a régi operátorkészlettel jobb eredményt ért el az algoritmus, mint az új, bővített operátorkészlettel. A kisebb adatsoroknál (Lighting2, Lighting7, OliveOil) ez könnyen lehet a kis méretből adódó zaj, ami rárakódik a mérésre. Ugyanakkor a FordB adatsornál nem ez a helyzet. Ott az operátor szintű túltanulás hatása figyelhető meg, amikor egy általunk definiált operátor túl jól írja le a tanítóhalmaz sajátosságait, és ezért az operátort használó modellek pontatlanabbak lesznek a teszthalmazon[9]. Sajnos ez ellen nem sokat lehet tenni, legjobb esetben is csak annyit tehetünk, hogy egy validációs halmazon méréseket végzünk különböző operátorkészletekkel és valamilyen heurisztika mentén megpróbáljuk megállapítani, hogy mely operátorok okozhatják egy adott probléma esetén túltanulás jelenségét. Szerencsére, mint az az adatokból is látszik, az operátor szintű túltanulás ennyi operátor mellett még ritka jelenség, és a pontosságot is csak minimálisan csökkenti.

Az egyes tanítási módok között az eltérés minimális, de az összes, heurisztikán alapuló módszer összességében jobbnak bizonyult az alap módszernél, ahol mindig az első legjobb vágást választottuk egy csomópontban. Az UCR adatokon az SM+ bizonyult a legjobbnak. Az UCR adatokon belül is megfigyelhető egy olyan trend, hogy az SM+ inkább ott volt jó, ahol kevés tanítóminta állt rendelkezésre egy-egy osztályból, azaz ahol a ShiftTree algoritmustól, mint modell alapú algoritmustól nem várjuk el, hogy nagyon jól teljesítsen. Az SM+ ezeken a helyeken növelte meg (főként) a pontosságot, ami nagyon értékes módszerré teszi ezt a tanítási módot. Az SM++ és az SM3+ a nagyobb tanítóhalmazzal rendelkező adatsorokon ért el jobb eredményeket. Ezek alapján a tanítási mód kiválasztását is problémafüggővé tehetjük: ha kevés tanítómintánk van osztályonként, de még elég ahhoz, hogy működjön a ShiftTree, akkor válasszuk az SM+ módot, ami a kis mintaszámból adódó zajra kevésbé érzékeny. Ha sok tanítómintánk van, akkor válasszuk az SM++ vagy SM3+ tanítási módok egyikét, ami a kis mintaszámból adódó zajra érzékeny ugyan, de ha sok minta áll rendelkezésünkre, akkor pontosabb modelleket hoz létre, mint az SM+.

Az eredményeket az 1-NN eredményeivel összehasonlítva a következőket láthatjuk. Egyrészt nagyságrendileg az euklideszi távolságot használó 1-NN nagyságrendileg a régi, szűk operátorkészletet használó ShiftTree-vel azonos eredményeket produkál, míg a DTW-t használó 1-NN a bővített operátorkészlettel dolgozó ShiftTree-hez hasonló pontosságokat ér el. Másrészt nagyon jól elkülöníthető, hogy mikor képes az 1-NN jobb eredményeket elérni a ShiftTree-nél. Ez akkor történik meg, ha az egy osztályra eső átlagos mintaszám a tanítóhalmazban kicsi. Ez azért van, mert a modell alapú módszerek, mint amilyen a ShiftTree is, akkor képesek igazán jól dolgozni, ha sok tanítóminta áll rendelkezésre, ami alapján általánosítani lehet, és létre lehet így hozni egy modellt, ami az adatsor általános tulajdonságait írja le. A példány alapú módszerek viszont kis adatmennyiség mellett is képesek pontosak lenni, mivel az egyes minták közötti közvetlen hasonlóságokat használják fel a címkézéshez, viszont emiatt képtelenek az általánosításra. Ez az oka annak, hogy pl. a Ford adatok esetében a régi operátorkészlettel dolgozó ShiftTree is jelentősen jobb eredményeket ér el, mint bármelyik távolságfüggvénnyel dolgozó 1-NN, viszont pl. a FaceAll probléma esetében mindkét 1-NN változat sokkal jobb, mint bármelyik tanítási móddal futtatott ShiftTree.

[9] A probléma bonyolult, mivel az operátorszintű túltanulás nem feltétlenül egy operátor miatt jelentkezik, mivel a ShiftTree egyes ágain az ESO szekvenciák együttesen határozzák meg a szem helyzetét és így a dinamikus attribútumokat.



7. ábra: ShiftTree és 1-NN az átlagos tanítómintaszám függvényében

A 7. ábra azt mutatja meg, hogy ha az eredmények összegzéséből kihagyjuk az adott küszöb feletti (balra), illetve alatti (jobbra) átlagos osztályonkénti tanítóminta számmal rendelkező adatsorokat, akkor hogyan alakul a ShiftTree és az 1-NN módszerek pontossága a küszöb értékének függvényében. Látható, hogy ha egyre nagyobb osztályonkénti átlagos tanítóminta számmal rendelkező adatsorokat tartunk csak meg, akkor a ShiftTree pontossága egyre inkább távolodik az 1-NN pontosságától. Bár minden módszer pontosabb lesz ilyenkor, a ShiftTree pontossága sokkal jobban megnő. Ha viszont csak egyre kisebb osztályonkénti átlagos tanítóminta számmal rendelkező adatsorokat vizsgálunk, akkor a ShiftTree pontossága jobban csökken, mint az 1-NN módszereké.

A 5. táblázat tartalmazza a többszörös (MM) tanítási módszerekkel elért eredményeket. A táblázat fejlécében zárójelben látható számok jelölik a korlátokat. Az első szám az összes modell számára adott felső korlát, míg a második (ha van) a csomópontonkénti vágások maximális száma. A korlátok úgy lettek beállítva, hogy a tanítás az összes adatsorra kivárható idő alatt lefusson, és a folyamat memóriaigénye se menjen 2GB fölé egyik esetben sem.

Megfigyelhető, hogy az eredmények jobbak, mint a sima SM módszer eredményei, ugyanakkor rosszabbak, mint bármelyik, heurisztikát is alkalmazó egyszerű módszer eredménye. Érdekes, hogy az MM és a heurisztikát is használó többszörös módszerek között nincsen túl nagy eltérés. Ennek az az oka, hogy ha a csomópontokban lekorlátozzuk a modellszámot, akkor azzal sok információt veszthetünk a fa magasabb szintjein, ami jelentősebb hatással van a modell jóságára, mint az alsó szinteken nyerhető plusz információ, amit azért nyerünk, mert később érjük el a globális modellkorlátot, mint az MM esetében.

Mindezeket figyelembe véve megállapítható, hogy a fejlesztett egyszerű tanítási módszerek mellett már nem érdemes az MM módszereket használni, mivel ésszerű korlátozások mellett nem pontosabbak, mint a heurisztikát használó SM módszerek. Ráadásul a futási idejük, és a modell mérete a sokszorosa az SM módszerekének. Emellett korlátozás nélkül nem is stabilak, különböző adatsorokon, operátorkészlettől függően, irreálisan sokáig futhatnak és irreálisan nagy modelleket hozhatnak létre. Ha korlátozás nélkül futtatjuk az MM módszereket, akkor, már amennyiben a módszer lefut, elérhetünk az SM módszereknél minimálisan jobb pontosságot, de a trade-off nem megfelelő. Ráadásul az 5. fejezetben bemutatott modell kombinálással sokkal rövidebb idő alatt sokkal pontosabb "többszörös" modelleket hozhatunk létre.

Adatsor	MM(1000)		MM+(2000,10)		MM++(2000,10)		MM3+(2000,10)	
	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság
50Words	247	54,29%	251	55,16%	238	52,31%	238	52,31%
Adiac	221	56,52%	227	58,06%	208	53,20%	208	53,20%
Beef	14	46,67%	16	53,33%	16	53,33%	14	46,67%
CBF	895	99,44%	874	97,11%	868	96,44%	868	96,44%
Coffee	25	89,29%	25	89,29%	25	89,29%	25	89,29%
ECG200	100	100,00%	100	100,00%	100	100,00%	100	100,00%
FaceAll	1103	65,27%	1110	65,68%	1109	65,62%	1109	65,62%
FaceFour	72	81,82%	73	82,95%	68	77,27%	67	76,14%
Fish	135	77,14%	132	75,43%	125	71,43%	127	72,57%
GunPoint	147	98,00%	136	90,67%	147	98,00%	147	98,00%
Lighting2	41	67,21%	41	67,21%	43	70,49%	43	70,49%
Lighting7	51	69,86%	49	67,12%	48	65,75%	48	65,75%
OliveOil	25	83,33%	21	70,00%	20	66,67%	20	66,67%
OSULeaf	138	57,02%	130	53,72%	130	53,72%	134	55,37%
SwedishLeaf	482	77,12%	500	80,00%	494	79,04%	494	79,04%
SyntheticControl	290	96,67%	281	93,67%	288	96,00%	281	93,67%
Trace	100	100,00%	100	100,00%	100	100,00%	100	100,00%
TwoPatterns	3985	99,63%	3995	99,88%	3998	99,95%	3987	99,68%
Wafer	6164	100,00%	6164	100,00%	6164	100,00%	6164	100,00%
Yoga	2466	82,20%	2493	83,10%	2497	83,23%	2498	83,27%
FordA	1233	93,41%	1241	94,02%	1237	93,71%	1237	93,71%
FordB	539	66,54%	545	67,28%	547	67,53%	558	68,89%
AE	308	83,24%	309	83,51%	307	82,97%	307	82,97%
UCR összes	**16701**	**89,78%**	**16718**	**89,87%**	**16686**	**89,70%**	**16672**	**89,62%**
Ford összes	**1772**	**83,19%**	**1786**	**83,85%**	**1784**	**83,76%**	**1795**	**84,27%**
Összes	**18781**	**89,00%**	**18813**	**89,15%**	**18777**	**88,98%**	**18774**	**88,97%**

5. táblázat: Többszörös tanítások összehasonlítása

4.4.2. Nyesés hatása

A 6. táblázat mutatja a három implementált nyesési módszerrel elért eredményeket. A tanításhoz az SM+ módot használtam. A táblázatnak hat oszlopa van. Az első négy oszlopban található eredmények úgy készültek, hogy a tanítóhalmazból rétegelt mintavételezéssel kivettem az idősorok 30%-át véletlenszerűen[10]. A megmaradt 70%-nyi adaton tanítottam, a kivett 30%-ot használtam validációs halmazként (amennyiben szükség volt rá), és az eredeti teszthalmazokat használtam a végleges pontosság lemérésére, mint a korábbi méréseknél. Ez a mérést adatsoronként 20-szor ismételtem meg, és a mérési eredmények átlagát vettem végleges eredménynek. A táblázat utolsó két oszlopa olyan módszereket tartalmaz, ahol nincs szükség validációs halmazra, így az egész tanítóhalmazt használtam fel a tanításhoz. A szignifikancia alapú nyesésnél a legjobbnak tűnő szignifikancia érték 0,001 volt, így ennél a módszernél ezt használom. Ez az érték elég szigorú nyesést eredményez.

A táblázat első négy oszlopát vizsgálva azt láthatjuk, hogy az egyszerű nyesés az egyetlen, ami jobbnak néz ki, mint a nyesés nélküli modell. Meg kell jegyezni viszont, hogy a mérési eredmények 20 mérés átlagaként állnak elő, és a mérési pontok szórása miatt ennyire kis eltérésnél nem mondhatjuk azt, hogy az egyszerű nyesés egyértelműen jobb a nyesés nélküli modellnél. A másik kettő, kifinomultabb, nyesési módszer érdekes módon rosszabbul teljesít. Az UCR adatokon ráadásul a szignifikancia alapú nyesés eredménye annyival alacsonyabb, mint a nyesés nélküli módszeré, hogy kijelenthetjük, hogy egyértelműen rosszabb.

[10] A 30%-nyi validációs halmaz méret bizonyult a kísérletek során optimálisnak.

Adatsor	Tanítóhalmaz 70%, validációs halmaz 100%, 20 mérés átlagolása								Nincs validációs halmaz			
	Nincs nyesés		Simple Pruning		Signi(0.001)		Complexity		Nincs nyesés		Signi(0.001)	
	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság	Találat	Pontosság
50Words	207,75	45,66%	204,85	45,02%	196,7	43,23%	207,45	45,59%	244	53,63%	236	51,87%
Adiac	186,9	47,80%	185,1	47,34%	183,45	46,92%	186,3	47,65%	220	56,27%	222	56,78%
Beef	11,95	39,83%	12,55	41,83%	11,9	39,67%	12,05	40,17%	17	56,67%	14	46,67%
CBF	812,25	90,25%	801,95	89,11%	812,25	90,25%	803,25	89,25%	848	94,22%	848	94,22%
Coffee	22,45	80,18%	22,7	81,07%	22,8	81,43%	23	82,14%	23	82,14%	23	82,14%
ECG200	100	100,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%
FaceAll	1009,65	59,74%	1011,3	59,84%	1002,5	59,32%	1017,8	60,22%	1108	65,56%	1108	65,56%
FaceFour	59,3	67,39%	60,1	68,30%	34,45	39,15%	42,15	47,90%	62	70,45%	54	61,36%
Fish	112,8	64,46%	112,8	64,46%	112	64,00%	113,65	64,94%	130	74,29%	130	74,29%
GunPoint	134,3	89,53%	134,3	89,53%	134,3	89,53%	123,95	82,63%	143	95,33%	143	95,33%
Lighting2	40,15	65,82%	40,3	66,07%	39,95	65,49%	40,7	66,72%	44	72,13%	43	70,49%
Lighting7	41,85	57,33%	41,15	56,37%	37,85	51,85%	40,8	55,89%	46	63,01%	45	61,64%
OliveOil	21,7	72,33%	21,15	70,50%	19,4	64,67%	19,95	66,50%	20	66,67%	20	66,67%
OSULeaf	130,05	53,74%	127,95	52,87%	131,8	54,46%	131,95	54,52%	136	56,20%	128	52,89%
SwedishLeaf	449,6	71,94%	447,95	71,67%	441,85	70,70%	449,25	71,88%	480	76,80%	478	76,48%
SyntheticControl	272,15	90,72%	273,95	91,32%	274,05	91,35%	271,6	90,53%	276	92,00%	276	92,00%
Trace	100	100,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%	100	100,00%
TwoPatterns	3991,1	99,78%	3991,1	99,78%	3991,1	99,78%	3977,05	99,43%	3994	99,85%	3994	99,85%
Wafer	6162,6	99,98%	6162,85	99,98%	6162,85	99,98%	6162,85	99,98%	6163	99,98%	6163	99,98%
Yoga	2277,65	75,92%	2297,05	76,57%	2245,35	74,85%	2286,55	76,22%	2559	85,30%	2558	85,27%
FordA	1204,15	91,22%	1216,05	92,13%	1202,55	91,10%	1203,35	91,16%	1229	93,11%	1232	93,33%
FordB	549,2	67,80%	549	67,78%	547,8	67,63%	549,95	67,90%	543	67,04%	541	66,79%
AE	294,65	79,64%	295,85	79,96%	294,3	79,54%	297,4	80,38%	287	77,57%	293	79,19%
UCR összes	**16144,2**	**86,79%**	**16149,1**	**86,81%**	**16054,55**	**86,31%**	**16110,3**	**86,61%**	**16713**	**89,85%**	**16683**	**89,68%**
Ford összes	**1753,35**	**82,32%**	**1765,05**	**82,87%**	**1750,35**	**82,18%**	**1753,3**	**82,31%**	**1772**	**83,19%**	**1773**	**83,24%**
Összes	**18192,2**	**86,21%**	**18210**	**86,30%**	**18099,2**	**85,77%**	**18161**	**86,06%**	**18772**	**88,96%**	**18749**	**88,85%**

6. táblázat: Nyesési módszerek eredményei

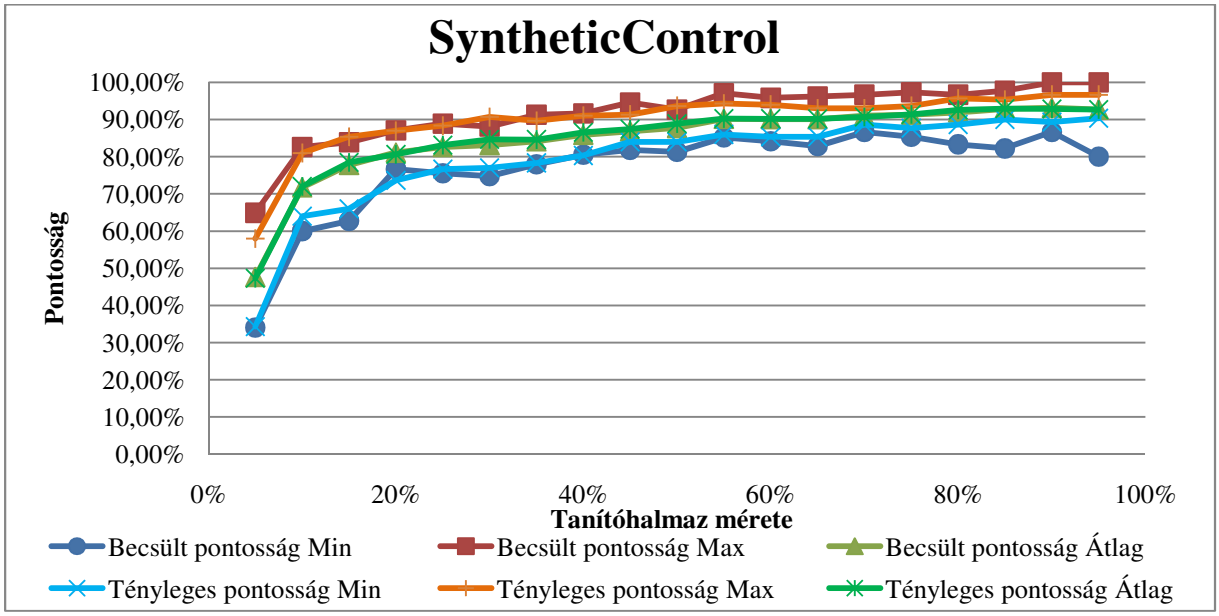
Ha csak a Ford adatokat nézzük, akkor viszont azt tapasztaljuk, hogy a nyesés alkalmazása megnövelte a pontosságot. Ennek két oka van: egyrészt a Ford problémákhoz az algoritmus nagy, terebélyes modellt hoz létre, másrészt a tanítóhalmaz elég nagy ahhoz, hogy a leválasztott validációs halmaz értelmesen használható legyen a nyesés során.

Ha megnézzük a táblázat utolsó két oszlopát, akkor egyrészt azt láthatjuk, hogy a szignifikancia alapú nyesés itt is rosszabbul teljesít, mint a nyesés nélküli modellezés. Hasonlítsuk össze viszont, a nyesési módszerek erejét leginkább demonstráló, Ford adatokon a legjobb nyesési eljárást az első négy oszlopból, és a nyesés nélküli modellezést (5. oszlop). Az eredmények azt mutatják, hogy jobban járunk, ha minél több adatot használunk fel a tanításhoz, és nem foglalkozunk a nyeséssel, mintha a tanítóhalmaz egy részét kihagyjuk a tanításból, és utólag próbáljuk optimalizálni a modellt nyeséssel. Mindez egyébként azt is jelenti, hogy a modell struktúra szinten, ezeken az adatokon, nem tanul túl.

4.4.3. Az algoritmus megbízhatósága

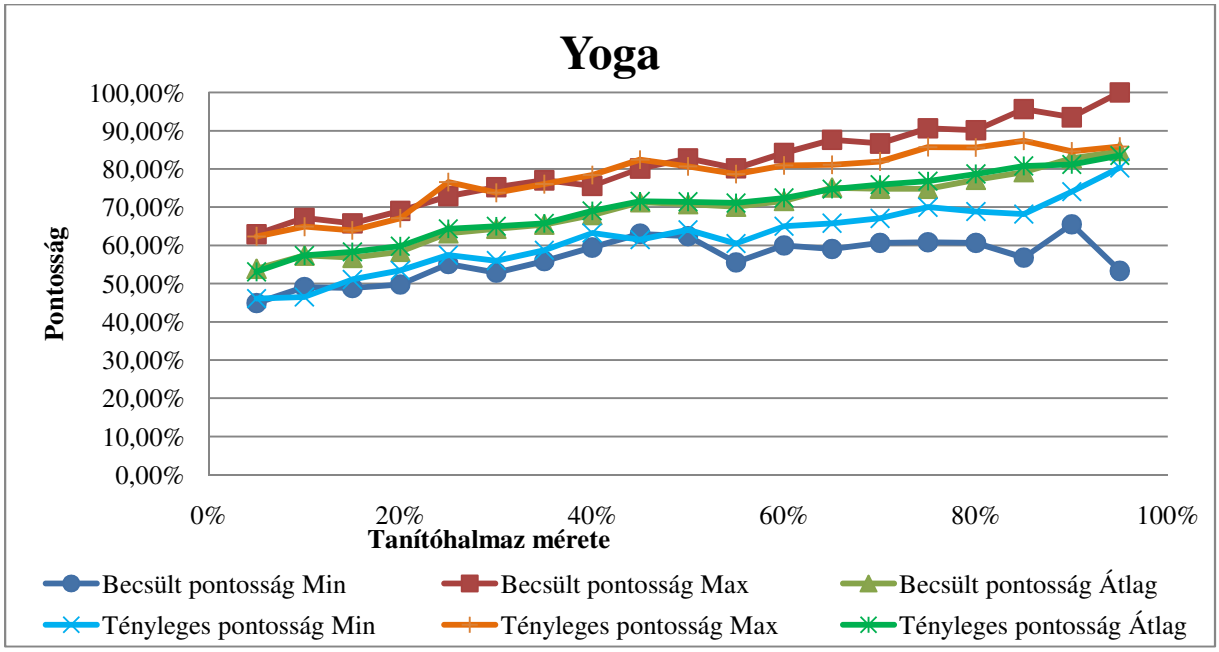
Az osztályozó algoritmusoknál az elsődleges teljesítménykritérium az, hogy mennyire pontosan címkéz. A gyakorlatban viszont nem elhanyagolható tulajdonság, hogy a tesztelt algoritmusunkon mért eredményeket mennyire fogadhatjuk el pontosnak. Nem jó, ha 95%-os pontosságúnak hitt modellt az új adatoknak mindössze a 80%-át tudja helyesen címkézni. Fontos megjegyezni, hogy ez a fajta megbízhatóság, vagy robosztusság nem csak az osztályozó, hanem az osztályozó és a megoldani kívánt probléma együttes tulajdonsága. Ugyanakkor kellően sok problémán tesztelve a módszert, állíthatjuk, hogy a módszer általában megbízhatóan, vagy megbízhatatlanul jelzi előre a pontosságot.

A ShiftTree robosztusságát lemérendő, a következő kísérletet hajtottam végre. A tanítóhalmazt két részre bontottam rétegelt mintavételezéssel: a halmaz X%-a alkotta a tényleges tanítóhalmazt, a maradék pedig a validációs halmazt. A tényleges tanítóhalmazt felhasználva, SM+ módszerrel tanítottam a ShiftTree-t, majd a validációs halmazon lemértem egy becsült pontosságot, a teszthalmazon pedig egy tényleges pontosságot. Ezt X minden értékére 20-szor elvégeztem. A 20 mérés eredményei közül vettem a minimumot, a maximumot és az átlagot mind a becsült, mind a tényleges pontosságoknál. Az alábbiakban néhány adatsorra megmutatom az így felrajzolható görbéket.



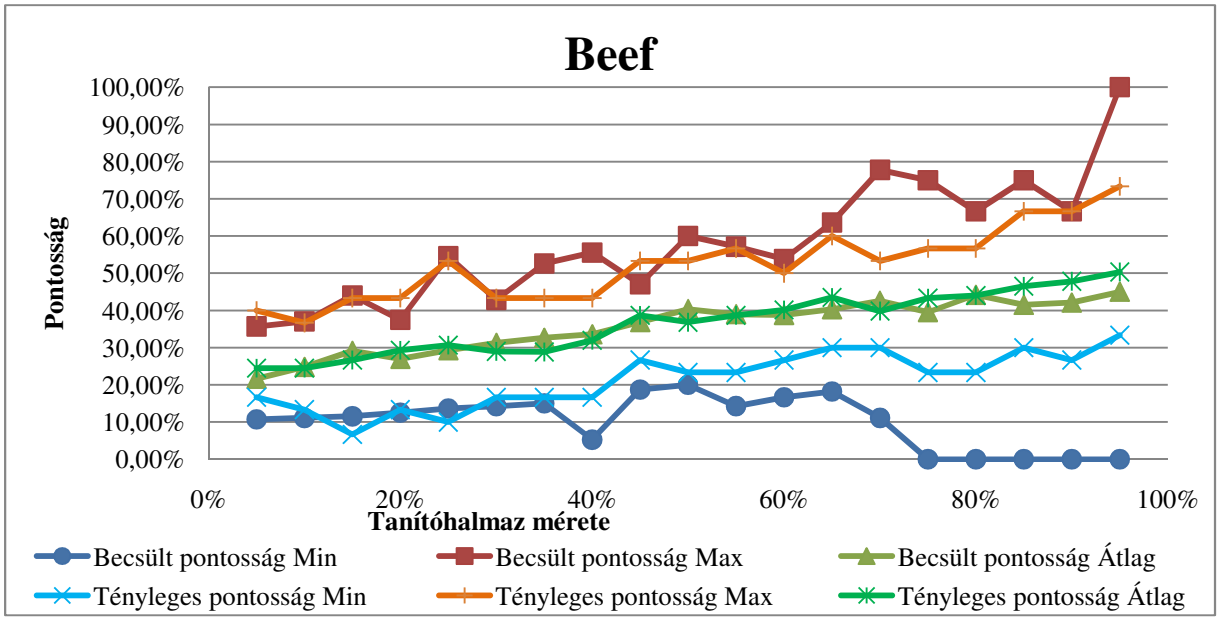
8. ábra: SyntheticControl robosztusság mérése

A 8. ábra a SyntheticControl adatsorhoz tartozó görbéket mutatja. Látható, hogy a becsült és a tényleges pontosságok a teljes mérési intervallumon együtt mozognak. A meglepő, hogy ez nem csak az átlagra, de a minimumra és a maximumra is igaz. Ez a jelenség az adatsorok többségénél jelentkezik, tehát úgy tűnik, hogy a ShiftTree eléggé robosztus.



9. ábra: Yoga robosztusság mérése

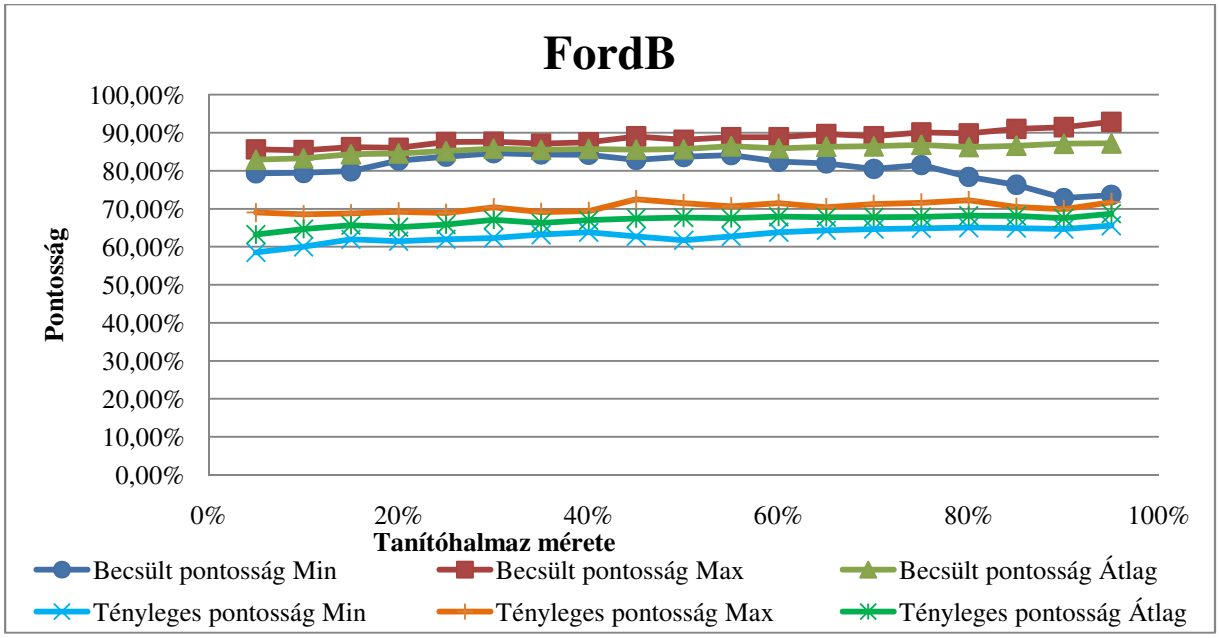
A Yoga adatsorhoz tartozó görbék (9. ábra) annyiban térnek el az előzőektől, hogy a tényleges pontosság minimuma és maximuma magas tanítóminta számnál közelít az átlaghoz, azaz az egyes mérések tényleges pontosságainak szórása csökken. A becsült pontossággal ez nem történik meg, sőt egy kicsit nő is a minimum és a maximum közötti távolság. A jelenségnek az az oka, hogy a tesztminta viszonylag nagy, a tanítóhalmaz pedig közepes. A mérési intervallum vége felé így a leválasztott validációs halmaz kicsi lesz, így annak az eredményei jobban fognak szórni, míg a tanítóhalmaz növekedése miatt a modell pontos lesz, így a nagymennyiségű tesztmintán elvégzett mérések eredményeinek szórása lecsökken. Az átlagok ugyanúgy egybe esnek, mint korábban, így itt is mondhatjuk, hogy a módszer megbízható. Viszont vegyük észre, hogy ez az eredmény kevésbé jó, mint amit a SyntheticControl adatsornál láttunk, mivel egy mérés során a becsült pontosság valószínűleg nem fog egybe esni a tényleges pontossággal. Ha viszont a modellünk pontosságát keresztvalidáció segítségével próbáljuk meg megbecsülni, akkor pontos eredményt kapunk.



10. ábra: Beef robosztusság mérés

Érdekes megfigyelni, hogy még nagyon kis méretű adathalmazok esetében is, mint amilyen például a Beef adatsor, egybe esik a becsült és a tényleges pontosság átlaga (10. ábra). Hiába óriási az eltérés, mind a pontosságok egyes mérési eredményei között, keresztvalidációval még mindig pontos képet kaphatunk az algoritmus teljesítményéről.

A 23 vizsgált adatsor közül egy olyan volt, ahol a becsült és a tényleges pontosság átlaga nem esett egybe, ez pedig a FordB probléma volt (11. ábra). Itt a becsült pontosság minimuma a mérési intervallumon végig a tényleges pontosság maximuma felett volt, annak ellenére, hogy az egyes mérési pontokban a mérések szórása kicsi. A FordB és a FordA feladat megegyezik, csak a FordB-nél zajosabbak az adatok. A becsült pontosságok mindkét problémánál hasonlóak, a tényleges pontosság viszont a FordB-nél 20-25 százalékponttal alacsonyabb. Ez azért van, mert a FordB feladat teszthalmaza jelentősebben eltér a tanítóhalmaztól, mint a FordA teszthalmaza annak a tanítóhalmazától. Amennyiben a teszthalmazt összevonjuk a tanítóhalmazzal és véletlenszerűen új teszthalmazt választunk, a jelenség nem jelentkezik.



11. ábra: FordB robosztusság mérése

4.4.4. Vak tesztek

A korábbi teszteknél a paraméterek optimalizálásához így vagy úgy, de felhasználtam valamennyi információt a teszthalmazból is. Hogy az algoritmus valódi képességeire fény derüljön, vak teszteket végzek a TSC adatokon. Ezek az adatok egy 2007-es verseny adatai. Mivel rendelkezésemre állnak a versenyen az indulók által elért eredmények, ezért a ShiftTree által elért eredményeket a versenybe illesztem, mintha plusz egy indulóként ez az algoritmus is részt vett volna a versenyben. A pontozás a következők szerint történt: ha egy problémán az algoritmus a legjobb volt, akkor 10 pontot kap, ha második volt, akkor 9-et és így tovább. Tízedik helyezés fölött nem jár pont. A versenyen 12 csapat adott be megoldást az összes problémára, a ShiftTree lesz a 13. induló.

Mivel az adatok tulajdonságai hasonlítanak az UCR adatokéhoz, az ott optimális beállításokat használom. Az operátorkészlet a szokásos, a VI. függelékben leírt bővített operátorkészlet. A tanítási mód SM+, nincs nyesés és a teljes tanítóhalmazt felhasználom a tanításhoz. Az operátorok paramétereit nem hangolom az egyes problémákhoz.

Adatsor	Találat	Pontosság	Helyezés	Pontok
TSC01	2043	87,12%	1	10
TSC02	956	92,91%	2	9
TSC03	817	50,43%	1	10
TSC04	1065	98,16%	1	10
TSC05	1231	89,20%	4	7
TSC06	107	34,74%	12	0
TSC07	800	80,40%	9	2
TSC08	485	63,82%	13	0
TSC09	399	66,39%	12	0
TSC10	765	80,27%	9	2
TSC11	836	73,40%	11	0
TSC12	7774	94,39%	1	10
TSC13	229	74,84%	3	8
TSC14	844	67,41%	13	0
TSC15	2416	62,92%	1	10
TSC16	734	85,25%	6	5
TSC17	846	90,38%	8	3
TSC18	219	39,82%	13	0
TSC19	1335	65,12%	13	0
TSC20	263	41,22%	11	0
Összesítve	**24164**	**78,24%**	**8**	**86**

7. táblázat: A ShiftTree eredményei a TSC adatokon (vak teszt)

A 7. táblázat tartalmazza a ShiftTree eredményeit. Mivel nem rendelkezek az összes résztvevő csapat és a szervező engedélyével, a résztvevők eredményeit nem teszem közzé, csak azt jelzem, hogy az egyes problémákon és összesítve hanyadik helyet ért volna el a ShiftTree. Az algoritmus összesítettben a 8. helyen végzett, ami önmagában is jó eredmény, tekintve, hogy az indulók feltehetőleg sokat optimalizált, kombinált módszerekkel dolgoztak, a ShiftTree-t pedig általános beállításokkal futtattam. Ráadásul a ShiftTree a 20 problémából 5 esetében az első helyen végzett, ami pontosan annyi első hely, amennyit az összetett győztes tudott szerezni. Itt is igaz, hogy a ShiftTree főként a nagyobb tanítóhalmazzal rendelkező adathalmazokon volt képes jó eredményeket elérni. Ha összesítjük a 20 probléma során eltalált idősorok számát, akkor a ShiftTree a nyertes módszerétől nem sokkal lemaradva a második legjobb eredményt éri el. Így annak ellenére, hogy az adatsorok tulajdonságai nem a ShiftTree-nek kedveztek, és optimalizált, kombinált módszerekkel szállt szembe, sikerült elég jó eredményeket elérnie.

5. Modellek kombinálása (forest eljárások)

Az adatbányászatban elterjedt gyakorlat, hogy egy problémára nem csak egy modell építünk, hanem több modell egyidejű alkalmazásával oldjuk meg a feladatot. Mivel az adatbányászati problémák bonyolultak, nem is várhatjuk, hogy egy modell tökéletesen le tudja írni a bemeneti adattömeg összes általános tulajdonságát, miközben figyelmen kívül hagyja az adatokon lévő zajt. Több modellnek együttesen nagyobb esélye van leírni a kívánt tulajdonságokat, miközben az egyes modellek esetleges túltanulása is kiátlagolódik a kombinálás során.

Kombinálhatjuk különböző algoritmusok modelljeit, de azt is megtehetjük, hogy egy adott algoritmus segítségével építünk több különböző modellt. Mivel a döntési fák viszonylag egyszerű struktúrák, és a tanításuk sem tart sokáig, nagyon elterjedt megoldás, hogy egy problémára több döntési fát is építsenek, amik halmazát döntési fa erdőnek (decision forest) neveznek.

Ebben a fejezetben két olyan eljárást (forest eljárást) mutatok be, aminek segítségével döntési fa erdőt hozhatunk létre akár ShiftTree-ből is. Ezekre ShiftForest eljárásokként hivatkozok. Az egyik ilyen módszer a gyakran használt, elterjedt, boosting, a másik egy saját fejlesztés, ami a keresztvalidáción alapul. A fejezet végén összehasonlítom a módszereket, és megvizsgálom, hogy mikor melyiket érdemes használni.

5.1. Boosting

A boosting [14] egy elterjedt módszer azonos típusú modellek kombinálására. A módszer nem csak azt mondja meg, hogy hogyan kombináljuk a modelleket, hanem azok építésébe is beleszól közvetetten, hogy a lehető legjobb összetett modellt kapjuk. Alapvetően az osztályozók kombinálására találták ki, de léteznek kiterjesztései más feladatokra is (pl. regresszió).

A sikeres kombinálás két feltétele az, hogy a kombinálandó modellek önmagukban is a lehető legpontosabbak legyenek, és az, hogy a modellek minél inkább eltérjenek egymástól [15]. Az előbbi nyilván algoritmus (és adatsor) függő. A boosting módszerek úgy dolgoznak, hogy próbálják az utóbbi feltételt teljesíteni, azaz minél eltérőbb modelleket kipréselni a tanítóalgoritmusból, de úgy, hogy azok pontossága ne legyen túl alacsony.

A legelső, és a mai napig gyakran használt, boosting eljárás az AdaBoost [16]. A módszer sematikus váza a következő:

- 1. Minden tanítóminta kapjon egy súlyt. Kezdetben ez legyen 1.
- 2. Normáljuk a súlyokat úgy, hogy az összegük 1 legyen.
- 3. Tanítsunk egy modellt úgy, hogy figyelembe vesszük a súlyokat.
- 4. Mérjük le a súlyozott osztályozási hibát a tanítóhalmazon.
- 5. A hiba alapján számoljunk egy modell súlyt.
- 6. Csökkentsük azon tanítóminták súlyát, amelyeket az adott modell helyesen osztályozott, és növeljük azokét, amelyeket nem. A változtatás mértéke a hiba nagyságának függvénye.
- 7a. Ha nem értük el a maximális iteráció számot, és az osztályozási hiba nem nulla, akkor folytassuk a 2. lépéstől.
- 7b. Ha elértük a maximális iteráció számot, vagy az osztályozási hiba nulla akkor álljunk le.

Az AdaBoost tehát úgy próbálja meg az adatsor jellemzőit több modellel megfogni, hogy a folyamatosan rosszul osztályozott pontoknak egyre nagyobb súlyt ad, így idővel az algoritmus

olyan modellt is fog építeni, ami ezeket, az eddig tipikusan rosszul osztályozott pontokat fogja jól osztályozni.

Az algoritmus kimenete több modell és az azokhoz tartozó modell súlyok. A címkézés ezután súlyozott szavazással történik, azaz minden modellnek a bemenetre adott válaszát akkora súllyal vesszük figyelembe, amekkora a modell súlya. A legtöbb súlyt kapó címke lesz a végleges válasz.

A módszert és az eddig megismert ShiftTree algoritmust több ponton is egyeztetni kell egymással. Először is a ShiftTree-nek át kell térnie a súlyok használatára, és ezeket a tanítás során figyelembe kell tudnia venni. Ez könnyen megoldható, ha a $PLabels$ függvény darabszám helyett súlyösszeget használ, azaz:

$$PLabels(l_i) = \frac{\sum_{n=1}^{N_{TR}} w_n I_{\{n|\langle \Theta_n, L_n \rangle,\ L_n = l_i\}}}{\sum_{n=1}^{N_{TR}} w_n}$$

Ez az átfogalmazás az algoritmusban sehol sem okoz gondot. Így a tanítás során, az optimális vágásoknál figyelembe vesszük az egyes idősorok súlyait is. A futási idő csökkentésére tett módosításokat (4.1. alfejezet) sem befolyásolja ez a módszer, mivel folytonos értékkészlettel sikerült bebizonyítanom, hogy a minimumok csak bizonyos helyeken fordulhatnak elő.

A második egyeztetendő dolog a tanítóhalmazon történő hiba mérés. A ShiftTree alapvetően a tanítóhalmaz szempontjából tökéletes modelleket hoz létre, azaz az osztályozási hiba a tanítóhalmazon alaphelyzetben nulla, így egy modell építése után mindig leállnánk, azaz a módszernek semmilyen hatása nem lenne. Ez a probléma megoldható a nyesés alkalmazásával. Bár a 4.4.2.-ben kiderült, hogy a nyesés a modell pontosságán valamennyit ront, a romlás nem akkora mértékű, hogy ellensúlyozza a modellkombináció miatt keletkező pontosságnövekedést. A 4.4.2. eredményei alapján a teljes tanítóhalmazon való tanulás előnyösebb, mint a validációs halmaz alapú utólagos nyesés, így egyértelmű, hogy a χ^2 alapú nyesést érdemes alkalmaznunk, mivel ez az egyetlen olyan nyesési eljárásunk, ami nem igényel validációs halmazt.

A harmadik megoldandó probléma a súlyok változtatásának és a modellsúly kiszámításának a kérdése. Az AdaBoost az alábbi képletet használja a M modellsúly kiszámítására és a w_i mintasúlyok változtatására, ahol err a modell (súlyozott) osztályozási hibája:

$$M = ln\left(\frac{1 - err}{err}\right)$$

$$w_i := \begin{cases} w_i \dfrac{err}{1 - err}, ha\ \hat{L}_i = L_i \\ w_i \dfrac{1 - err}{err}, ha\ \hat{L}_i \neq L_i \end{cases}$$

A képletekből is látható, hogy a módszer akkor működik helyesen, ha $err < 0{,}5$. Ellenkező esetben a modell súlya negatív lesz, és a súlyokat is a rossz irányban változtatjuk. A 0,5-ös hibaarány megegyezik egy véletlenszerű osztályozó várható hibájával kétosztályos esetben. Mivel az AdaBoost eredetileg kétosztályos osztályozók kombinációjára lett kitalálva, ez a követelmény a hibára egy elvárható kritérium volt. Láthattuk, hogy a ShiftTree minden vizsgált problémán 0,5-ös hibaarány alatt teljesített, ráadásul itt nem a teszthalmazon vett hibáról van szó, hanem a tanítóhalmazon vettről, ami a fa kialakítása miatt jóval alacsonyabb. Ezek ellenére még sem tűnik biztonságosnak a módszer alkalmazása, mivel pl. egy 50 osztályt tartalmazó probléma (50Words) esetén, (ahol a véletlenszerű osztályozó várható hibája 0,98) nem szerencsés, ha ilyen szigorú hibakritériumot állítunk. Ezért nem az AdaBoost-tal, hanem egy másik boosting variánssal dolgozok.

5.1.1. SAMME

A SAMME [17], azaz (Stagewise Additive Modeling using a Multi-class Exponential loss function) egy olyan boosting variáns, amit kifejezetten többosztályos osztályozási problémákhoz fejlesztettek ki. A szerzők szerint a módszerük jobb eredményeket ér el többosztályos osztályozási problémák esetén, mint a klasszikus AdaBoost. Lényegi változás a súlyok számításában van, méghozzá az alábbi módon:

$$M = ln\left(\frac{1-err}{err}\right) + ln(N_C - 1)$$

$$w_i := \begin{cases} w_i, ha\ \hat{L}_i = L_i \\ w_i \frac{1-err}{err}(N_C - 1), ha\ \hat{L}_i \neq L_i \end{cases}$$

Azaz az alkalmazott függvények kiegészültek egy $ln(N_C - 1)$ taggal (a mintasúly szorzója felírható e^M alakban). Ez a tag úgy módosít a maximális hibára vonatkozó kritériumon, hogy a maximális hiba, amit az osztályozó véthet megegyezik a véletlenszerű osztályozó várható hibájával. Minden értelmes osztályozótól elvárható, hogy ezt a kritériumot teljesítse.

Egy másik változás az, hogy a helyesen címkézett minták súlya közvetlenül nem változik, csak a többi elem súlyának növelése miatt a normalizáláskor csökken. Ez igazából egy technikai részlet, és csak azt határozza meg, hogy milyen gyorsan változzon meg a jól és rosszul osztályozott pontok súlyainak aránya. A fentiek alapján a ShiftTree-ben alkalmazott boosting eljárás a 4. algoritmussal írható le.

Bemenet: Egy címkézett idősor halmaz $TR = \{\langle \Theta_n, L_n \rangle\}_{n=1}^{N_{TR}}$ és a hozzá tartozó szemek $Cursors = \{C_n\}_{n=1}^{N_{TR}}$, és K maximális iteráció szám
Kimenet: $< R_k, M_k >_{k=1}^{K}$ K darab csomópont, ami ugyanennyi modellt reprezentál, mint gyökér elem, illetve a modellekhez tartozó súlyok

procedure ShiftForestBoost$(TR, Cursors, K)$

$err \leftarrow 1$
$k \leftarrow 0$
$W = \langle w_n = 1 \rangle_{n=1}^{N_{TR}}$
while $err > 0$ **and** $k < K$ **do**
  $k \leftarrow k + 1$
  $W \leftarrow$ Normalize(W)
  $R_k \leftarrow$ BuildShiftTree$(TR, Cursors, W)$
  $err \leftarrow 0$
  for n = 1 **to** N_{TR} **do**
    $C_n \leftarrow 1$
    $\hat{L}_n =$ShiftTreeLabel$(R_k,\ \Theta_n,\ C_n)$
    if $\hat{L}_n \neq L_n$ **then**
      $err \leftarrow err + w_n$
    end if
    $C_n \leftarrow 1$
  end for
  $M_k \leftarrow ln\left(\frac{1-err}{err}\right) + ln(N_C - 1)$
  for n = 1 **to** N_{TR} **do**
    if $\hat{L}_n \neq L_n$ **then**
      $w_n \leftarrow w_n \frac{1-err}{err}(N_C - 1)$
    end if
  end for
end while
return $< R_k, M_k >_{k=1}^{K}$
end procedure

4. algoritmus: ShiftForest boosting alapon

5.2. XV módszer

Az XV módszer egy nagyon egyszerű, általam a ShiftTree-hez fejlesztett modell kombináló eljárás. Az eljárás általános, tehát bármilyen osztályozóval használható.

A módszer a keresztvalidációból indul ki. A lényege az, hogy csak a tanítóhalmaz egy részén végezzük el a tanítást, egy másik részét megtartjuk a modell ellenőrzésére (validációs halmaz). Az ellenőrzés során kapunk egy becsült pontosságot a modellre. Ezt a pontosságot rendeljük hozzá a modellhez, mint súlyt. A tanítást ezután megismételjük a tanítóhalmaz egy másik részhalmazán. A címkézés a boostinghoz hasonlóan itt is súlyozott szavazással történik.

Bár a módszer nagyon egyszerű, mégis egy helyes modellkombinálási technika, mivel a tanítóhalmazból vett különböző minták nagy valószínűséggel egymástól eltérő modelleket fognak eredményezni, az algoritmus megbízhatósága miatt pedig a validációs halmazon mért becsült pontosságok jól jellemzik a modellek fontosságát.

Az XV módszernek két paramétere van, az egyik (S) határozza meg a validációs halmaz méretét az eredeti tanítóhalmaz méretéhez képest, a másik (K) pedig az iterációk számát adja meg. A módszert írja le az 5. algoritmus:

Bemenet: Egy címkézett idősor halmaz $TR = \{\langle \Theta_n, L_n \rangle\}_{n=1}^{N_{TR}}$ és a hozzá tartozó szemek $Cursors = \{C_n\}_{n=1}^{N_{TR}}$, K maximális iteráció szám, és S a validációs halmaz mérete az eredeti tanítóhalmaz méretéhez képest
Kimenet: $< R_k, M_k >_{k=1}^{K}$ K darab csomópont, ami ugyanennyi modellt reprezentál, mint gyökér elem, illetve a modellekhez tartozó súlyok

procedure ShiftForestXV($TR, Cursors, K, S$)
1: **for** $k = 1$ **to** K **do**
2: $\langle TR_k, VA_k \rangle \leftarrow$ StratifiedSplit(TR, S)
3: $R_k \leftarrow$ BuildShiftTree($TR, Cursors, W$)
4: $hit \leftarrow 0$
5: **for all** $\langle \Theta_i, L_i \rangle \in VA_k$ **do**
6: $C_i \leftarrow 1$
7: $\hat{L}_i =$ShiftTreeLabel(R_k, Θ_i, C_n)
8: **if** $\hat{L}_i = L_i$ **then**
9: $hit \leftarrow hit + 1$
10: **end if**
11: $C_n \leftarrow 1$
12: **end for**
13: $M_k \leftarrow \frac{hit}{N_{VA_k}}$
14: **end for**
15: **return** $< R_k, M_k >_{k=1}^{K}$
end procedure

5. algoritmus: ShiftForest XV alapon

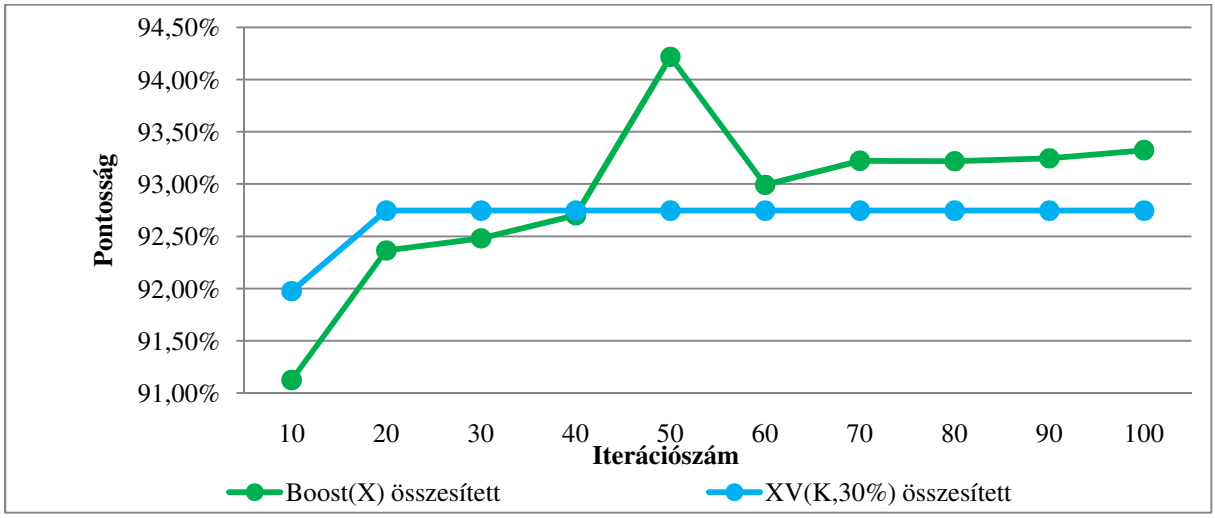
A StratifiedSplit(TR, S) metódus rétegelt mintavétel segítségével véletlenszerűen szétosztja a TR tanítóhalmaz mintáit két halmazt (TR_k és VA_k) létrehozva. Az osztályok eloszlása a két halmazban közel azonos. VA_k mérete az eredeti TR halmaz méretének S-szerese.

5.3. Forest módszerek vizsgálata

Ebben az alfejezetben megvizsgálom, hogy a korábban bemutatott forest módszerek mennyire hatékonyak, hogyan érdemes beállítani a paramétereiket. Emellett kísérleteket végzek annak kiderítésére is, hogy milyen esetekben érdemes boostingot és milyen esetekben érdemes XV-t használni. Az alfejezet végén a legjobb kikísérletezett beállítással vak teszteket végzek a TSC adatok segítségével.

5.3.1. Alap kísérletek

A két forest eljárásnak nincs sok paramétere, mindkettőnél meg kell adni az iteráció számot, és az XV-nél ezen kívül a validációs halmaz méretét is. Ezeknek a paramétereknek a helyes megválasztása viszont fontos lehet a kombinált modell pontosságának szempontjából. A paraméterek optimális értékének megtalálására kísérleteket végeztem az UCR és a Ford adatbázisokon. A használt operátorkészlet a VI. függelékben leírt bővített operátorkészlet, a tanítási mód az UCR adatokon legjobbnak bizonyuló SM+. A boosting során szignifikancia alapú nyesést használok szigrorú nyeséséi kritériummal (szignifikancia szint 0,001). Az XV módszer során nem használok nyesést.



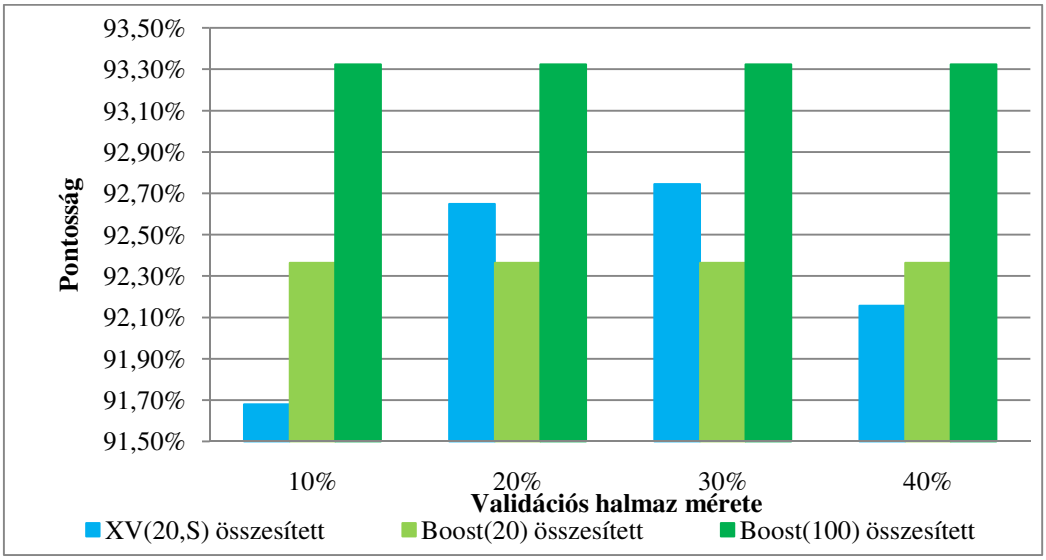
12. ábra: Boosting és XV pontossága az iterációk számának függvényében

A 12. ábra mutatja azt, hogy hogyan függ a boosting módszerrel előállított ShiftTree modellekből álló erdő (ShiftForest) pontossága az iterációk számától. Az adatok a vizsgált adatsorokon elért eredmények összegeként állnak elő. Referenciának felvettem az XV módszer eredményét is az adott iteráció szám mellett, $S = 30\%$ mellett. A boosting grafikonjában $K = 50$-nél egy kiugrás látható, de ez nagy valószínűséggel csak az adatokra jellemző zaj miatt van. Ha ettől a kiugrástól eltekintünk, azt látjuk, hogy az iterációk számának növelésével a pontosság is növekszik, természetesen egyre kisebb mértékben.

Az is megfigyelhető, hogy az XV módszer már $K = 20$-nál eléri a maximumát, és tovább nem nő. Azt tapasztaltam, hogy bármilyen értelmes S-re $K = 20$ körül az XV módszer már eléri, vagy megközelíti a maximális értékét, és további iterációk beiktatásával az eredmények nem változnak. Ez összességében nem meglepő, mivel $S = 30\%$ mellett $K = 20$-nál minden tanítópontot várhatóan 14-szer fogunk felhasználni a modell építéséhez, ami elég magas szám ahhoz, hogy a különféle tulajdonságú tanítóminták sokféle kombinációja előforduljon a tanítóhalmazokban. Így a további iterációk már lényegében ugyanazokat a modelleket adják vissza, mint amik már korábban is előálltak, azaz csak az egyes modellek súlya változik.

A 13. ábra mutatja be, hogy hogyan változik az XV módszer pontossága attól függően, hogy az S paramétert mennyire állítjuk. A korábbi tapasztalatok alapján az iterációszámot 20-ra állítottam, mivel ott az XV módszer már eléri a maximumát értelmes S értékekre. Referenciaként felvettem a boosting pontosságát 20, illetve 100 iteráció mellett. Az ábráról az olvasható le, hogy az XV módszer túl kicsi és túl nagy validációs halmaz méret mellett is rosszul teljesít. Ez nem meglepő, mivel ha a validációs halmaz mérete túl kicsi, akkor az azon lemért eredmények szórása elég nagy lesz, így az egyes modellekhez rendelt súlyok (becsült pontosság értékek) nem fogják jól jellemezni az adott modell jóságát, így a kombináció sem lesz optimális. Túl nagy validációs halmaz esetén pedig nem jut elég pont a tanítóhalmazba,

így a ShiftTree által épített modellek lesznek rosszak, amit utána hiába tudunk pontosan kiértékelni, rossz modellekből nem tudunk pontos kombinált modellt építeni.



13. ábra: XV pontossága a validációs halmaz méretének függvényében

Végül optimális paraméternek az $S = 30\%$ adódott, azaz az eredeti tanítóhalmaz 30%-át érdemes használni validációs halmazként.

Adatsor	Tanítóhalmaz mérete	Teszthalmaz mérete	Boost(100)		XV (20,30%)		SM+	
50Words	450	455	339	74,51%	318	69,89%	244	53,63%
Adiac	390	391	273	69,82%	255	65,22%	220	56,27%
Beef	30	30	20	66,67%	15	50,00%	17	56,67%
CBF	30	900	848	94,22%	874	97,11%	848	94,22%
Coffee	28	28	23	82,14%	23	82,14%	23	82,14%
ECG200	100	100	100	100,00%	100	100,00%	100	100,00%
FaceAll	560	1690	1353	80,06%	1318	77,99%	1108	65,56%
FaceFour	24	88	63	71,59%	81	92,05%	62	70,45%
Fish	175	175	158	90,29%	145	82,86%	130	74,29%
GunPoint	50	150	143	95,33%	145	96,67%	143	95,33%
Lighting2	60	61	38	62,30%	44	72,13%	44	72,13%
Lighting7	70	73	61	83,56%	55	75,34%	46	63,01%
OliveOil	30	30	20	66,67%	23	76,67%	20	66,67%
OSULeaf	200	242	185	76,45%	174	71,90%	136	56,20%
SwedishLeaf	500	625	569	91,04%	530	84,80%	480	76,80%
SyntheticControl	300	300	276	92,00%	293	97,67%	276	92,00%
Trace	100	100	100	100,00%	100	100,00%	100	100,00%
TwoPatterns	1000	4000	3994	99,85%	3994	99,85%	3994	99,85%
Wafer	1000	6164	6163	99,98%	6164	100,00%	6163	99,98%
Yoga	300	3000	2729	90,97%	2694	89,80%	2559	85,30%
FordA	3601	1320	1281	97,05%	1277	96,74%	1229	93,11%
FordB	3636	810	612	75,56%	606	74,81%	543	67,04%
UCR összes	**5397**	**18602**	**17455**	**93,83%**	**17345**	**93,24%**	**16713**	**89,85%**
Ford összes	**7237**	**2130**	**1893**	**88,87%**	**1883**	**88,40%**	**1772**	**83,19%**
Összesítés	**12634**	**20732**	**19348**	**93,32%**	**19228**	**92,75%**	**18485**	**89,16%**

8. táblázat: Forest módszerek eredményei adatsoronként

Az optimálisan paraméterezett módszerek eredményeit érdemes adatsoronként is összehasonlítani egymással, illetve az optimális egyszeres ShiftTree modellel. Az iterációk számát a boostingnál $K = 100$-ra állítottam, mivel az 50-nél lévő csúcs esetleg rossz irányba

befolyásolhatná a méréseket. Az iterációk további növelésével még nőne a pontosság, igaz egyre lassabban, de $K = 100$-nál még elfogadható a módszer futási ideje is.

A 8. táblázat tartalmazza az eredményeket. Látható, hogy mindkettő forest módszer sokkal jobb eredményeket produkál, mint a legjobb egyszeres modell, és pontosan ez volt a célunk ezeknek a módszereknek a bevezetésével. Ha visszagondolunk a korlátozott MM módszerekre, amikről 1000-2000 modellt tartalmaztak, de nem voltak jobbak, mint az SM+ tanítás, akkor a javulás még szembetűnőbb.

A táblázatnak van néhány olyan sora, ahol az alap SM+ tanítással előállított modell és a boosting által előállított ShiftForest pontossága megegyezik. Az ilyen esetekben a kezdeti modellen elvégzett nyesés, hiába volt az kifejezetten szigorú, nem nyesett le egy ágat sem, így a tanítóhalmazon az osztályozási hiba nulla volt, és a módszer egy modell létrehozása után leállt. Ilyen esetekben nem érdemes a boostingot alkalmazni.

Azoktól az esetektől eltekintve, amikor a boosting nem használható, mindössze két esetben jobb az XV a boostingnál (FaceFour, Lighting2). Mindkét adatsorban közös, hogy kicsi a tanítóhalmaz elemszáma. Ellentétben tehát azzal, amikor a 4.4.-ben azokat a tanítóhalmazokat tekintettük kicsinek, ahol az osztályonkénti átlagos tanítópontok száma van egy bizonyos határ érték alatt, az XV és a boosting közötti döntésnél az a fontos, hogy a tanítóhalmaz elemeinek a száma hogyan viszonyul egy határértékhez. A fenti adatokon ez a határérték 60, ami fölött boostingot érdemes használni (amennyiben az működik), alatta, illetve 60 méretű tanítóhalmaznál pedig XV-t. Erre a szabályra is van egy ellenpélda az adatokban (Beef), de jó kezdeti heurisztikának tűnik.

5.3.2. Forest eljárások vak tesztje

Az UCR adatokon kikísérletezett optimális beállításokkal végrehajtottam egy vak tesztet a TSC adatokon. A tesztelés menete teljesen analóg azzal, amikor a legjobb egymodelles módszert teszteltem (4.4.4. pont). A 12 versenyző csapat mellé 13-nak vettem a legjobb ShiftForest módszert. A ShiftTree ebben az összehasonlításban nincs benne.

Adatsor	Találat	Pontosság	Helyezés	Pontok
TSC01	2176	92,79%	1	10
TSC02	975	94,75%	2	9
TSC03	822	50,74%	1	10
TSC04	1059	97,60%	1	10
TSC05	1224	88,70%	4	7
TSC06	139	45,13%	11	0
TSC07	792	79,60%	10	1
TSC08	593	78,03%	12	0
TSC09	458	76,21%	12	0
TSC10	769	80,69%	9	2
TSC11	852	74,80%	11	0
TSC12	8037	97,58%	1	10
TSC13	288	94,12%	1	10
TSC14	925	73,88%	13	0
TSC15	2773	72,21%	1	10
TSC16	699	81,18%	6	5
TSC17	891	95,19%	5	6
TSC18	274	49,82%	7	4
TSC19	1864	90,93%	9	2
TSC20	413	64,73%	11	0
Összesítve	**26023**	**84,26%**	**5**	**96**

9. táblázat: ShiftForest vak teszt

A beállítások az előző pontban kikísérletezett optimális beállítások, azaz 100 iterációs boosting, amikor nem működik, vagy a tanítóhalmaz mérete kisebb vagy egyenlő, mint 60 minta, akkor pedig 20 iterációs XV, 30%-os validációs halmaz mérettel.

A vak teszt eredményei a 9. táblázatban láthatóak. Összességében a módszer az ötödik helyen végzett volna, egy ponttal lemaradva a negyedik helyről. A 20 feladatból hatot ez a módszer oldott meg a legjobban, ami több, mint bármely másik módszer esetében. Az összesített pontossága 84,26%, ami szintén jelentősen magasabb, mint bármely induló összesített pontosságának értéke.

A 4.4.4. pont méréseivel összehasonlítva látszik, hogy nem változott jelentősen, hogy mely adatsorokon tud a módszer jó eredményeket elérni. A kevés osztályonkénti átlagos tanítómintával rendelkező adatsorokra modell alapú módszereket nem nagyon tudunk optimalizálni, akármit is csinálunk. Ha ezeknél az adatsoroknál egy egyszerű, DTW alapú 1-NN-t használnánk, a többinél pedig a legjobb ShiftForest eljárást, akkor ez a vegyes módszer egyértelműen az első három között végezne.

A dolog nehézsége az, hogy nem tudjuk pontosan megmondani, hogy mely adatsorok olyanok, amin a ShiftTree-t (ShiftForest-et) nem érdemes alkalmazni. Egyfelől alkothatunk heurisztikákat, amik a tanítóhalmaz mérete, és az osztályonkénti átlagos tanítóminta szám függvényében megbecsülik, hogy érdemes-e a ShiftTree-t alkalmazni. Ugyanakkor ez nem lesz egy stabil módszer, mivel az alkalmazhatóság nem csak ettől függ, hanem az adatsor tanulhatóságától is. Van egy elég széles sáv, amin belül hol a ShiftTree, hol az 1-NN teljesít jobban adatsortól függően.

Egy másik megközelítési mód az, hogy az algoritmus becsült pontossága alapján próbáljuk eldönteni, hogy alkalmazható-e a ShiftTree. Itt az is problémát okoz, hogy a becsült pontosságok átlaga csak az alap algoritmus pontosságát becsli jól, a ShiftForest módszerekét nem, azok pontossága valamennyivel mindig magasabb ennél a becsült értéknél. A másik probléma, hogy egy adott értékből nem állapítható meg, hogy mennyire erős az algoritmus egy adott problémán más algoritmusokhoz képest. Pl. a TSC15 adatsoron elért 72,21%-os érték nagyon jónak számít, mások meg sem közelítik, de a TSC07-en elért 79,6%-os eredmény nem túl jó.

A kettő heurisztika kombinálásával viszont nagy valószínűséggel megállapítható lehet, hogy érdemes-e egy adott problémára a ShiftTree-t alkalmazni.

6. Címkézési konfidencia

A 4.4. alfejezetben láthattuk, hogy a ShiftTree kellően nagy pontossággal képes megoldani a idősor-osztályozási feladatot. Az 5. fejezetben mutatott két forest módszer segítségével pedig elértük nagyjából azt a szintet, amit ezzel az operátorkészlettel pontosság szempontjából elérhető. Viszont néhány esetben a pontosság mellett az is fontos, hogy a modell pontosan meg tudja mondani, hogy mennyire biztos az általa adott válaszban, azaz mekkora az adott minta címkézésének konfidenciája. Az alapvető ShiftTree algoritmus minden esetben 100%-ig biztos a dolgában, így viszont amikor téved, akkor nagyot téved.

Ebben a fejezetben megvizsgálok néhány lehetséges konfidencia számítási módszert, amiket a ShiftTree-nél alkalmazni lehetne. A különböző módszereket összehasonlítom. A fejezet végén bemutatom azt, hogy hogyan lehet az osztályozási konfidenciákat egy a gyakorlatban fontos, de sokszor elhanyagolt feladat, az on-line tanulás, megoldására használni.

6.1. Csomópont-és útvonal konfidencia

Döntési fáknál az osztályozási konfidenciát általában a címkézési útvonal végén lévő levél tulajdonságai alapján definiálják. Ezt én csomópont vagy levél konfidenciának nevezem. Az elképzelés lényege, hogy biztosak vagyunk benne, hogy az osztályozandó adat a megfelelő levél csomópontba került, azaz csak azzal kapcsolatban vagyunk bizonytalanok, hogy a levélhez melyik címke mekkora valószínűséggel tartozik. Lehetséges levél konfidencia értékek:

- Osztály aránya a levélben: A legelterjedtebb osztályozási konfidencia döntési fáknál. Egy címke konfidenciája a hozzá tartozó minták relatív gyakorisága a levélben. Ezzel azt közelítjük, hogy mekkora a valószínűsége, hogy egy, a levélbe kerülő, minta címkéje a vizsgált osztályhoz tartozik. Pontatlan akkor, ha kis méretű levelek vannak a fán, azaz a levelekbe kevés tanítópont jut el. A ShiftTree-nél csak akkor használható, ha valamilyen nyesési eljárást alkalmazunk. Jelöljük ezt a módszert $Conf_A$-val. Értéke az $Node$ levélben:

$$Conf_A(l_i) = \frac{\sum_{n=1}^{N_{TR_{Node}}} I\{l_n = l_i\}}{N_{TR_{Node}}}$$

- Osztályból mennyi jut el a levélbe: A probléma egy másféle megközelítése. Azt mondja, hogy akkor vagyunk biztosak a címkézésben, ha az adott osztályból sok jutott el az adott levélbe. A visszaadott konfidencia érték a levélben a vizsgált osztályhoz tartozó minták száma és az adott osztály mintáinak a száma a teljes tanítómintában. Azt a valószínűséget közelíti, hogy egy adott címkéjű mintapont milyen valószínűséggel jut el ebben a levélbe. Nagy méretű tanítóhalmazok esetén bizonyos leveleknél túl alacsony lehet az értéke. Jelöljük ezt a módszert $Conf_B$-vel. Értéke a $Node$ levélben:

$$Conf_B(l_i) = \frac{\sum_{n=1}^{N_{TR_{Node}}} I\{l_n = l_i\}}{\sum_{n=1}^{N_{TR}} I\{l_n = l_i\}}$$

- Az előző kettő kombinációja: Érdemes lehet az előző kettő értéknek valamilyen kombinációját venni, például a szorzatukat. Ha magas ez a szorzat, az azt jelenti, hogy a csomópont közel homogén, és az adott osztályhoz tartozó címkéjű pontok többsége az adott csomópontba kerül. Szorzat helyett lehet súlyozott átlagot is használni, de figyelni kell a két összetevő esetleges nagyságrendi eltérésére.

Az útvonal konfidencia saját fejlesztés, az ötlet lényege az, hogy egy adott címkére adott konfidencia értéke ne csak a levél csomópont tulajdonságaitól függjön, hanem a bejárt címkézési útvonal összes csomópontjának tulajdonságaitól. Ez a megoldás sokkal bizonytalanabbnak tekinti a modellt, ami feltehetőleg közelebb áll a valósághoz, mint amit a levél konfidenciáknál feltételezünk.

Az osztályozás során a minta végigmegy egy útvonalon a fában. Minden egyes csomópontban van valamekkora konfidencia érték az összes lehetséges osztályra. Ezek a konfidenciák bármilyen csomóponti konfidencia képlet alapján adódhatnak. Ezeket valamilyen súlyozás szerint összegezzük, ez lesz az adott címkéhez tartozó konfidencia a minta esetén.

Az osztályozás kimenete az a címke, amelyikhez a legnagyobb konfidencia érték tartozik, és visszaadjuk ezt a konfidencia értéket is. Úgy is tekinthetünk erre a módszerre, mint egyfajta dinamikus utónyesésre, ami a csomópontok egy adott mintaeloszlása esetén bizonyos ágakat lenyes a fából. Viszont ha a minták eloszlása megváltozik, akkor a nyesés is megváltozik. A módszer további előnye a ShifTree-nél, hogy nem igényel előzetes nyesést az alkalmazása semmilyen belső konfidencia használata esetén, ellentétben a levél konfidenciával. Néhány lehetséges módszer:

- Csomóponti konfidenciák átlagolása: Az útvonal csomópontjai által adott konfidenciák egyszerű átlaga. Veszélye, hogy ha a tanítóhalmaz kiegyensúlyozatlan (bizonyos osztály mintáiból arányaiban kevés van a többi osztály mintáihoz képest), akkor a magas szinteken lévő csomópontok miatt a ritka osztályt a többiek elnyomhatják.
- Csomóponti konfidenciák súlyozott átlagolása: Az útvonal csomópontjai által visszaadott konfidenciákat valamilyen súlyokkal összegezzük. A súlyokat érdemes úgy megválasztani, hogy magasabb szinteken alacsonyabbak legyenek.
- Csak a többségi osztályok figyelembe vétele: Az összegzésnél csak az adott csomópontban a többségi osztályhoz tartozó konfidenciát vesszük figyelembe, a többi osztályhoz tartozó konfidenciát nullának tekintjük.

6.2. Eredmények konfidenciákkal

A konfidenciás mérésekhez a VI. függelékben leírt bővített operátorkészletet és SM+ tanulási módot használtam. Összehasonlítottam az előző alfejezetben leírtakból létrehozható összes lehetséges módszert. Ez három csomópont konfidenciát jelent és kilenc módszert jelent (mindhárom összegzési módszert mindhárom csomóponti konfidenciával). A súlyozott átlagolásnál a csomópont szintjét használtam fel súlyként, így a levélhez közeledve egyre nagyobb súllyal vesszük figyelembe a csomópontokat. Nyesést csak ott használtam, ahol szükséges, azaz csak a csomóponti konfidenciáknál, amikor a többségi osztály arányát (is) vizsgáljuk a levélben. A használt nyesésé a szokásos szignifikancia alapú nyesés 0,001-es szignifikancia értékkel.

A vizsgált mérőszámok a pontosság és az AUC voltak. Mivel az AUC többosztályos problémákra nem értelmezhető, mint egy mérőszám, ezért megmértem minden osztályra külön, mintha az lenne a pozitívnak nevezett osztály, és az összes többi a negatív, majd vettem a mért értékek közül a minimumot, az átlagot és a maximumot, és ezekkel viszonylag jól jellemezhetőek a módszerek.

Adatsor	Csomóponti, többségi gyakorisága, nyesett				Csomóponti, többségi idejutásának gyakorisága				Útvonal, súlyozott, többségi gyakorisága				Útvonal, súlyozott, csak többségi, többségi gyakorisága			
	Találat	MinAUC	AvgAUC	MaxAUC	Találat	MinAUC	AvgAUC	MaxAUC	Találat	MinAUC	AvgAUC	MaxAUC	Találat	MinAUC	AvgAUC	MaxAUC
50Words	233	0,4322	0,8052	0,9972	244	0,3240	0,6961	0,9985	239	0,2602	0,6403	0,9416	236	0,3598	0,7379	0,9879
Adiac	224	0,5777	0,8526	0,9989	223	0,4375	0,7932	0,9989	222	0,4196	0,6672	0,8845	220	0,4855	0,7517	0,9606
Beef	14	0,5799	0,7097	0,9653	17	0,6910	0,8576	0,9653	17	0,7153	0,8500	0,9653	17	0,7188	0,8542	0,9653
CBF	848	0,9225	0,9584	0,9983	848	0,9225	0,9584	0,9983	848	0,9569	0,9706	0,9956	848	0,9542	0,9689	0,9956
Coffee	23	0,8641	0,8705	0,8769	23	0,8769	0,8833	0,8897	23	0,8769	0,8833	0,8897	23	0,8769	0,8833	0,8897
ECG200	100	0,9863	0,9894	0,9924	100	0,9861	0,9891	0,9922	100	0,9861	0,9891	0,9922	100	0,9861	0,9891	0,9922
FaceAll	1114	0,5418	0,8718	0,9890	1108	0,3793	0,7903	0,9824	1113	0,5080	0,8678	0,9904	1117	0,6912	0,8697	0,9913
FaceFour	54	0,7022	0,8381	0,9618	62	0,6613	0,8334	0,9575	62	0,7649	0,8778	0,9721	62	0,7500	0,8719	0,9721
Fish	128	0,7825	0,8929	0,9947	130	0,6297	0,7726	0,9691	130	0,7161	0,8775	0,9721	130	0,7556	0,8713	0,9680
GunPoint	143	0,9603	0,9604	0,9605	143	0,9752	0,9780	0,9808	143	0,9752	0,9780	0,9808	143	0,9752	0,9780	0,9808
Lighting2	43	0,7073	0,7091	0,7110	44	0,6867	0,6886	0,6905	44	0,7008	0,7029	0,7051	44	0,7008	0,7029	0,7051
Lighting7	45	0,6169	0,7827	0,9714	46	0,5883	0,8088	0,9786	46	0,4701	0,7564	0,9786	45	0,5124	0,7646	0,9786
OliveOil	20	0,7986	0,8484	0,8810	20	0,7917	0,8403	0,8730	20	0,6587	0,8124	0,8750	20	0,7788	0,8324	0,8750
OSULeaf	136	0,6976	0,7986	0,9314	136	0,4791	0,5760	0,6791	136	0,6225	0,8165	0,9346	132	0,6233	0,8074	0,9340
SwedishLeaf	476	0,7333	0,9010	0,9958	481	0,4993	0,7532	0,9349	477	0,7514	0,8773	0,9950	467	0,6375	0,8751	0,9950
SyntheticControl	276	0,9080	0,9602	0,9863	276	0,8595	0,9414	0,9863	276	0,8724	0,9542	0,9912	276	0,8874	0,9546	0,9917
Trace	100	0,9932	0,9936	0,9942	100	0,9932	0,9936	0,9942	100	0,9932	0,9936	0,9942	100	0,9932	0,9936	0,9942
TwoPatterns	3994	0,9985	0,9992	0,9998	3994	0,9990	0,9995	0,9998	3994	0,9978	0,9992	0,9998	3994	0,9985	0,9994	0,9998
Wafer	6163	0,9998	0,9999	1,0000	6163	0,9998	0,9999	1,0000	6163	0,9998	0,9999	1,0000	6163	0,9998	0,9999	1,0000
Yoga	2558	0,8778	0,8778	0,8778	2559	0,1969	0,1969	0,1970	2582	0,8862	0,8862	0,8862	2525	0,8830	0,8833	0,8835
FordA	1206	0,9355	0,9355	0,9356	1220	0,1260	0,1261	0,1262	1241	0,9734	0,9735	0,9737	1232	0,9660	0,9690	0,9721
FordB	531	0,6608	0,6614	0,6620	530	0,4440	0,4441	0,4442	559	0,7484	0,7486	0,7489	527	0,7287	0,7391	0,7496
UCR összes	**16692**	**0,7840**	**0,8810**	**0,9542**	**16717**	**0,6988**	**0,8175**	**0,9033**	**16735**	**0,7566**	**0,8700**	**0,9472**	**16662**	**0,7784**	**0,8795**	**0,9530**
Ford összes	**1737**	**0,7981**	**0,7985**	**0,7988**	**1750**	**0,2850**	**0,2851**	**0,2852**	**1800**	**0,8609**	**0,8611**	**0,8613**	**1759**	**0,8473**	**0,8541**	**0,8608**
Összes	**18429**	**0,7853**	**0,8735**	**0,9401**	**18467**	**0,6612**	**0,7691**	**0,8471**	**18535**	**0,7661**	**0,8692**	**0,9394**	**18421**	**0,7847**	**0,8772**	**0,9446**

10. táblázat: Nyesési módszerek összehasonlítása

A 10. táblázat tartalmazza az eredmények közül az érdekesebbeket (nem közlöm, mind a 12 módszer összes eredményét). A csomóponti konfidenciát használó módszerek közül a $Conf_A$ módszer érte el a legjobb eredményt. A találatok száma a nyesés miatt alacsonyabb, mint a sima SM+ tanítású modellé. Ugyanakkor ha megvizsgáljuk az AUC értékeket, látható, hogy néhány extrém adatsortól eltekintve (50Words, Adiac, FaceAll és Beef), egész jó értékeket sikerült elérni. A csomóponti módszerek közül AUC-ra a legrosszabb a $Conf_B$ módszer lett. A többi módszerben sem tűnt hasznosnak a $Conf_B$ módszer használata, sem önmagában, sem a $Conf_A$-val együtt.

Az útvonali konfidenciát használó módszerek közül a szintszámmal növekvő súlyú módszerek rendre felülmúlták a súlyozást nem használó társaikat. Az a változat, ami csak a többségi osztály konfidenciáját veszi figyelembe az egyes csomópontokban, egy kicsit jbb AUC értéket ért el az UCR adatokon, mint az, amelyik az összes konfidenciát figyelembe veszi. Az utóbbi módszer viszont a Ford adatokon ért el jobb AUC eredményeket. Ráadásul ez a módszer egy kicsivel pontosabb modelleket eredményezett, mint a sima SM+ tanítás, azaz sikerült egy olyan nyesési módszert találni, ami nem igényel validációs halmazt, és minimálisan javít a modellen. Igaz a javulás nem jelentős.

Igazából problémafüggő, hogy mely módszert érdemes használni. A pontosság értékek miatt általános esetben az útvonalfüggő, a csomópontokban minden osztály konfidenciáját felhasználó, $Conf_A$ konfidenciával dolgozó módszert javasolt használni. Amennyiben a célunk az AUC növelése, kisebb méretű adatsorok esetén érdemes a csomópontokban csak a többségi osztály konfidenciáját figyelembe venni, vagy levél konfidencián alapuló módszert használni.

6.3. On-line tanulás

On-line tanulás alatt azt értjük, hogy a modellünk reagál az általa megismert új adatokra, és megfelelően módosítja magát. Ezt csak akkor tudja megtenni, ha az osztályozandó adatnak ismertté válik a címkéje. Tipikus esetei a módszernek a felhasználó visszajelzésein alapuló adaptív modellek. Például egy gesztusfelismerő, aminek expliciten megmondható, hogy nem jól ismerte fel az adott gesztust.

Az on-line tanulás lényege, hogy nem azt csináljuk, hogy minden egyes új adat beérkezésekor azt hozzávesszük a tanítóhalmazhoz, és nulláról elkezdjük felépíteni az új modellt, mivel erre általában nincs idő és/vagy erőforrás (főleg, ha gyakran érkezik új adat). Inkább a már meglévő modellünket módosítjuk valamilyen módon. Ez a módszer persze nem fog annyira pontos modellt eredményezni, mint egy teljes újratanítás, ezért sokszor nem foglalkoznak vele. Ugyanakkor a gyakorlatban használt rendszerekben nagyon fontos a megléte.

A 6.1. alfejezetben említettem, hogy az útvonal konfidenciát használó módszerek felfoghatóak adaptív utónyesési eljárásként, amik igazából nem is nyesik le a fa ágait, csak úgy manipulálnak a konfidenciákkal, hogy a gyakorlatban néhány levél elérhetetlen lesz. Viszont ha a csomópontokban az egyes osztályokhoz tartozó pontok száma megváltozik, akkor megváltoznak a konfidenciák is, és így az elérhető ágak is. Tehát egy alapvető on-line tanulási módszerhez elég annyit tennünk, hogy egy útvonal konfidenciás módszert használunk, és amennyiben visszajelzés érkezik bármely osztályozott pont címkéjéről, akkor az osztályozás során bejárt útvonalának minden csomópontjában megfelelő súllyal módosítjuk a minta címkéjéhez tartozó számlálót[11].

A módszer finomítható azzal, hogy ha egy levél csomópont már nagyon nem homogén, akkor azt az egy csomópontot a tanuló algoritmus segítségével kifejtjük. Ez nem minden esetben

[11] Ez természetesen megtehető levélkonfidenciával is, de a modell jóval lassabban fog reagálni, és sokkal instabilabb is lesz.

lehetséges, mert meg kell hozzá őrizni az összes olyan idősort, ami ebbe a levélbe került. Viszont nem biztos, hogy az adott eszközön rendelkezünk az ehhez szükséges tárhellyel vagy a kifejtéshez szükséges számítási kapacitással.

6.3.1. On-line tanulás hatékonysága

Az on-line tanulást a pontosság alapján legjobbnak bizonyuló módszerrel megvalósítottam. A tesztek során a tanítómintán felépítettem a modellt, majd elkezdtem osztályozni a tesztminta idősorait. Kétféle tesztet futtattam: az első tesztben minden osztályozás közben módosítottam a címkézés útvonalán a csomópontokban az osztályok eloszlását. A második tesztnél csak akkor végeztem el a módosítást, ha az osztályozó kimenete nem egyezett meg a az idősor tényleges címkéjével. Ez a teszt azt modellezi a felhasználót, aki csak hibás döntés esetén jelez vissza a rendszernek. Az első tesztnek egy másik variánsát is elvégeztem, amiben az osztályozott minták nagyobb súlyt kapnak, azaz jobban befolyásolják a csomópontok tulajdonságait. A tesztelés során azt is megvizsgáltam, hogy mi történik, ha egy tesztmintát többször átfuttatunk az osztályozón. Az első utáni iterációban kapott eredmények nem összehasonlíthatók a korábbiakkal, mivel ilyenkor a modell már a tesztmintákból is használt fel információt. A kísérlet viszont jó arra, hogy egyrészt az on-line tanulás hatását megmutassa, másrészt annak szimulálására, hogy mi történik, ha egy adott eloszlás szerint érkeznek az idősorok a rendszerünk bemenetére egy hosszabb idejű működés alatt.

A teszteket a Ford adatokon végeztem el, mivel csak olyan esetekben van értelme ezt a módszert tesztelni, amikor nagy, szerteágazó modellünk, és viszonylag kevés osztályunk van. Mivel kompakt modellek esetén nem történik semmi változás, sok osztály és kevés tanítópont esetén pedig a modell összevissza változik, így a tesztminták sorrendjétől nagyon függ az elért pontosság.

Adatsor	Iteráció	On-line tanítás nélkül		On-line tanítás az összes ponton		On-line tanítás csak a helytelenül osztályozottaknál		On-line tanítás az összes ponton, 100-szoros súllyal	
FordA	1. iteráció	1241	94,02%	1241	94,02%	1238	93,79%	1233	93,41%
	2. iteráció			1242	94,09%	1240	93,94%	1237	93,71%
	3. iteráció			1242	94,09%	1241	94,02%	1237	93,71%
	4. iteráció			1239	93,86%	1238	93,79%	1237	93,71%
	5. iteráció			1238	93,79%	1235	93,56%	1238	93,79%
	6. iteráció			1238	93,79%	1241	94,02%	1238	93,79%
	7. iteráció			1238	93,79%	1243	94,17%	1238	93,79%
	8. iteráció			1238	93,79%	1246	94,39%	1238	93,79%
FordB	1. iteráció	559	69,01%	563	69,01%	559	69,01%	564	69,14%
	2. iteráció			568	70,12%	562	69,38%	592	73,09%
	3. iteráció			572	70,62%	561	69,26%	598	73,83%
	4. iteráció			579	71,48%	544	67,16%	598	73,83%
	5. iteráció			582	71,85%	551	68,02%	598	73,83%
	6. iteráció			584	72,10%	552	68,15%	599	73,95%
	7. iteráció			585	72,22%	556	68,64%	601	74,20%
	8. iteráció			588	72,59%	559	69,01%	601	74,20%

11. táblázat: On-line tanítás eredmények

Az eredmények a 11. táblázatban láthatóak. Korábban már láthattuk, hogy a két Ford adatsor között egy lényegi különbség, hogy a FordB problémánál a teszthalmaz nagyon eltér a tanítómintától, míg a FordA-nál nem (lásd 4.4.3.), így a FordA adatsoron 90% feletti pontosságot érünk el, míg a FordB-nél jóval alacsonyabbat. Látszik, hogy az on-line tanítás olyan esetben hatékony, mint ami a FordB-nél is fennáll. Ilyenkor az eltérő tulajdonságú teszthalmaz megváltoztatja a modell tulajdonságait, és így a tesztmintához hasonló

tulajdonságú idősorokat az idő előrehaladtával egyre pontosabban fogunk tudni osztályozni. Gyorsít a javuláson az, ha az ilyen módosításoknak nagyobb súlyt adunk.

Olyankor, amikor a tesztminta idősorai hasonló tulajdonságokkal rendelkeznek, mint a tanítóminta idősorai, a hatás általában nem pozitív. Amennyiben az összes osztályozott idősor módosít a modellen, és a modellünk alaphelyzetben is pontos volt, nem igazán fog az megváltozni, ami nem meglepő, hiszen a tesztpontok erősítik az eredeti modellünket, ahelyett, hogy megváltoztatnák. Kisebb változások persze vannak, amiket a nem jól osztályozott pontok okoznak, de ezek inkább csak minimálisan csökkentik a modellünk teljesítményét.

Amennyiben csak a rosszul osztályozott pontok esetében van visszajelzésünk, az eredmények egy kicsit hektikusabbak. A FordA probléma esetében hullámzik a teljesítmény, majd végül magasabb lesz, mint ami eredetileg volt. Ez azért van, mert a tanítómintához hasonló tesztminta esetén a hibásan osztályozott (eltérő tulajdonságú) idősorok módosítják a modellt, ami így előbb pontosabb lesz, majd a kiugró tulajdonságú idősorok miatt egy kicsit pontatlanabb lesz. A folyamat végén beáll a modell egy adott szintre.

A FordB esetén, ahol a teszthalmaz eltér a tanítómintáktól, ez a módszer nem tűnik konvergensnek, mivel a tesztmintának egy része az egyik, és kb. egy ugyanakkora része a másik irányba próbálja módosítani a modellt.

Összességében tehát elmondható, hogy ha úgy érezzük, hogy a bejövő adataink tulajdonságai az idő előrehaladtával folyamatosan és jelentősen változnak, akkor érdemes olyan on-line tanítást megvalósítani, ahol az összes osztályozott minta módosítja a modellt, és érdemes a változtatásnak magas súlyt adni. A modell struktúrájának változatlansága miatt nem kell attól tartani, hogy emiatt az eredeti modellünk elromlana. Ha viszont a bejövő adatok lassan, vagy nem túl jelentősen változnak, akkor csak a rosszul osztályozott mintákat érdemes felhasználnunk az on-line tanulás során.

6.4. Konfidenciával kapcsolatos lehetséges fejlesztések

Ez a fejezet egy rövid betekintést kívánt nyújtani az osztályozási konfidenciával kapcsolatos világba és az azzal kapcsolatos fejlesztésekbe. A fejezet végén megemlítem azokat az irányokat, ami mentén a témát mélyebben érdemes megvizsgálni.

Érdemes megvizsgálni a konfidencia és a forest módszerek kapcsolatát. Kezdeti kísérletek azt mutatják, hogy a konfidencia használata nem ront, de nem is javít a többszörös modelleken. Viszont ha van konfidenciánk, akkor a forest módszereken is jelentősen változtathatunk. A boosting során a rosszul osztályozott pontok súlyváltoztatásánál egyértelmű, hogy felhasználhatjuk a konfidenciát, mint a módosítás erősségét szabályozó tényezőt. Érdemes viszont azt is megvizsgálni, hogy kétosztályos esetben nem a címkét vesszük az osztályozó kimenetén, hanem a konfidenciát, amit a negatív osztály esetében egyből kivonunk. Így a nulla és egy címkéket egy nulla és egy közötti számmal becsüljük. Ebben az esetben viszont használhatunk regressziós feladatokra optimalizált boostin eljárásokat.

Érdemes lehet bevezetni egy idősor konfidenciának nevezett konfidenciát az eddigiek mellé, helyett. Ezt együttesen alakítaná az osztályozandó idősor és a modell. Ez az érték azt mondja meg, hogy mennyire biztos az, hogy az adott idősor jó helyen van. Ha visszagondolunk az SM+ módszernél használt heurisztikára, akkor felírhatjuk, hogy az idősor dinamikus attribútuma mennyire tér el a vágási határtól. Ha közel van hozzá, akkor lehet, hogy csak a véletlen folytán kerül az adott oldalra. Szintén megvizsgálható, hogy a dinamikus attribútum értéke mennyire illik bele az idősorral azonos osztályba tartozó idősorok attribútumának eloszlásába. Ezek alapján megmondható, hogy az adott idősorra az adott osztályozási útvonal mennyire ad biztos címkézést.

7. Egy idősor-osztályozási feladat megoldása ShiftTree-vel

Ebben a fejezetben röviden bemutatom, hogy hogyan lehet egy új idősor-osztályozási feladatot megoldani a korábban bemutatott módszerek segítségével. A fejezet célja annak demonstrálása, hogy a ShiftTree és a kapcsolódó módszerek könnyen használhatóak idősor osztályozásra. Az elemzés nem teljes, mivel célom nem annak a kiderítése, hogy mi a legnagyobb elérhető pontosság, hanem azt megmutatni, hogy néhány egyszerű lépés segítségével is jó modelleket tudunk létrehozni.

7.1. A feladat ismertetése

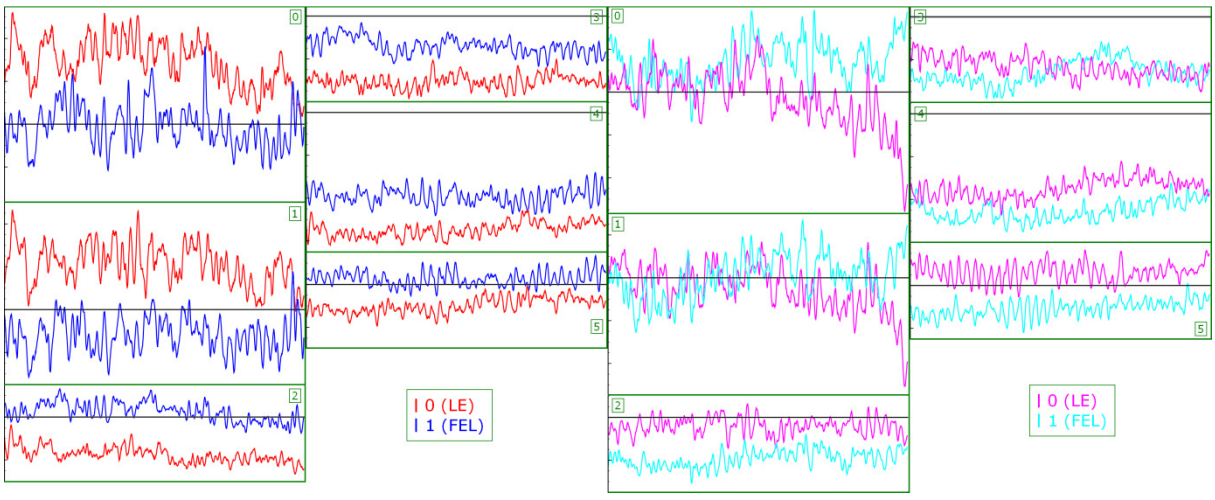
A feladat a 2003-as BCI (Brain-Computer Interface) verseny [17] első feladata. A leírás alapján [18] a kísérletben a tesztalany fejére 6 elektródát rögzítettek, mindet az agy szenzoros (az érzékelésért felelős) területein. Minden kísérletben vagy felfele, vagy lefele kellett mozgatnia egy kurzort a számítógép képernyőjén a gondolatai segítségével. Amennyiben az egyik csatornán az aktuális potenciálérték pozitív volt, a kurzor lefele indult, ha negatív volt, akkor felfele. A tesztalanynak minden esetben jelezték, hogy a lefele vagy a felfele mozgás-e a cél, valamint visszajelezték a kérdéses csatornán az agyhullámainak az alakját. A kísérlet során 3,5 másodpercig rögzítették az adatokat 256Hz –es mintavételi frekvenciával. A kísérlet két napon keresztül zajlott, és szerencsétlen módon a tanítóhalmazba nagyrészt olyan adatok kerültek, amiket az első napon rögzítettek, a teszthalmazba pedig csak olyanok, amiket a második napon rögzítettek.

7.2. Adatok elemzése

Az idősor fontosabb tulajdonságai a következők:

- Változók száma: 6
- Idősorok hossza: 896
- Osztályok száma: 2 (FEL és LE)
- Tanítóhalmaz mérete: 286
- Teszthalmaz mérete: 293

Az idősor tulajdonságai alapján úgy gondoljuk, hogy a ShiftTree alkalmas a probléma megoldására, mivel osztályonként elégséges számú tanítóminta áll rendelkezésünkre. Emellett az idősorok hossza is olyan tartományban van, ahol láttuk korábban, hogy a ShiftTree működik.



14. ábra: BCI adatsor osztályai

A 14. ábra mindkét osztályra két-két példát mutat. A világos és sötétkék színnel jelölt idősorok a FEL osztályhoz, a piros és rózsaszínnel jelöltek a LE osztályhoz tartoznak. Az egyes példákat direkt úgy választottam ki, hogy látszódjon, hogy nem egyértelmű az idősorok szétválaszthatósága. (A példák kiválasztása egy alap ShiftTree modell alapján történt, a négy példa között van igaz pozitív, igaz negatív, hamis pozitív és hamis negatív.)

7.2.1. Szakértői tudás bevitele a modellbe

Valamennyi előzetes tudásunk van a problémáról, ezzel modellezem a szakértői tudást, amit egy valódi problémánál egy igazi szakértő be tud építeni a modellekbe az operátorokon keresztül. Az itt leírtak nem tényleges szakértői tudáson alapulnak, és csak demonstrációs célokat szolgálnak.

Egyrészt tudjuk, hogy a különböző frekvenciájú agyhullámok különböző tevékenységekhez kötődnek, minél nagyobb a frekvencia, annál éberebb/koncentráltabb a megfigyelt alany. Próbáljuk meg figyelembe venni az agyhullámok frekvenciáját, illetve az attól való eltéréseket. Méréseink alapján két lokális maximum között átlagosan 24 egységnyi idő telik el a kapott adatokban. Ezért érdemes az operátoraink paramétereit 24 és annak többszöröseire, illetve 12-re (a periódus fele) állítani. Érdemes olyan operátorokat létrehozni, ami képesek kihozni az eltéréseket. Ez megoldható például olyan összetett operátorokkal, ahol egy/fél periódusnyi ugrás után megkeressük a lokális maximumot/minimumot, stb. Az ezen megfontolások alapján elkészített operátor struktúra a VI. függelékben CBI operátorkészlet címszó alatt megtalálható.

Ezen kívül tudjuk a leírásból az elektródák elrendezését. Láthatjuk, hogy a standard C3 és C4 mérési pontok környékén két mérési pont van. Érdekes lehet ezeknek az egymáshoz közel lévő mérési pontokban mért jelnek valamilyen kombinációját vizsgálni. A C3 és C4 pont illetve az A1-Cz és A2-Cz pontok szimmetrikusan helyezkednek el, azaz ugyanott vannak, csak az egyik a jobb, a másik a bal agyféltekén. Ezeknek valamilyen kombinációja szintén érdekes lehet. Ezekhez a kombinációkhoz VVO-kat használunk. A pontos operátorok megtalálhatóak a VI. függelékben.

7.2.2. Modellezés és kiértékelés

Egy versenyen a teszthalmaz címkéi titkosak, de általában a beküldött megoldások eredményei a beküldés után nem sokkal ismertté válnak a beküldő számára. A napi beküldések száma viszont sokszor limitált. Ezért nem építhetek akárhány modellt, mindössze négy változatot fogok készíteni, és ezeket értékelem ki.

Az első változat a VI. függelék bővített operátorkészletével dolgozik, SM+ tanulást használ, és semmi mást. A második változat a problémára létrehozott új operátorkészletet használja a VVO-k nélkül, és SM+ tanulást. A harmadik változat az új operátorokat használja, beleértve a VVO-kat is. A negyedik változat egy XV eljárással készített ShiftForest, 20 iterációval és 30% méretű validációs halmazzal.

A modellek pontossága rendre 77,76%, 78,5%, 79,52% és 85,67%. A VVO-kat is használó modellben fontos szerepet tölt be az a változó, ami az A1-Cz és A2-Cz pontokon mért változóknak az eltéréseként áll elő. Látható, hogy a "szakértői" tudás bevitele még ilyen minimális esetben is jelentősen növeli a modell pontosságát. Emellett itt is használhatók a forest módszerek a pontosság növelésére. Ezzel az egyszerű elemzéssel jól megmutatható, hogy milyen könnyen lehet használni a ShiftTree-t: már alap operátorkészlettel is kellően pontos eredményeket ad, de ha van valamilyen előzetes tudásunk a problémáról, akkor azt könnyen beleépíthetjük a modellbe, ami így pontosabbá válik.

8. Kitekintés

Ebben a fejezetben röviden ismertetek kettő olyan területet, ahová a ShiftTree algoritmust ki lehetne terjeszteni. Adatok hiányában eredményeket nem tudok bemutatni, de úgy gondolom, hogy mindenképpen fontos, hogy lássuk, hogy a ShiftTree nem csak egy idősor-osztályozó, hanem egy olyan algoritmuscsalád alapja, ami sok probléma megoldására alkalmas lehet.

8.1. Jelek felismerése adatfolyamokban

Az utóbbi néhány évben kezdett népszerű kutatási területté válni az úgy nevezett stream-mining, azaz az adatfolyamok adatbányászata. Ennek oka az, hogy az elmúlt 10 évben a szenzorok előállítási költsége jelentősen csökkent, így olcsón és egyszerűen lehet folyamatos méréseket végezni. Az így előálló adatok adatfolyamok, amik egy vagy több érték végtelen hosszú sorozatát jelentik. Az adatfolyamok feldolgozásakor többféle feladat is elképzelhető. Az egyik ilyen feladat az, hogy ismerjük fel, ha valami történik, azaz a jelfelismerés. Ez a feladatcsoport tovább bontható: feladat lehet, hogy ismerjük fel, hogy a rendszer a szokásostól eltérően viselkedik, és egy másik feladat az, hogy ismerjük fel, hogy a streamben előre meghatározott jelek fordulnak elő.

Az utóbbi feladat egyik izgalmaz alkalmazási módja a számítógép-agy interfészekben (BCI-kben) lehetséges. Ezek az eszközök úgy működnek, hogy a felhasználó agyáról valamilyen módon információkat gyűjtünk (az ára miatt az EEG tűnik manapság úgy, mint megvalósítható alternatíva), majd ezt továbbítjuk egy feldolgozó szoftvernek. Szoftveres oldal feladata, hogy ha a felhasználó egy adott dologra gondol, akkor azt a rendszer azonosítani tudja, és ennek megfelelően az ehhez a gondolathoz tartozó parancsot hajtsa végre. Például ha a felhasználó szeretné megnézni az emailjeit, akkor elképzel maga előtt egy borítékot, a gép pedig elindítja az email kliens. De ugyanez a feladat egy olyan, mobiltelefonokon futó, gyorsulásmérőt használó gesztusfelismerő, ami állandóan fut, de csak akkor hajt végre parancsot, ha a felhasználó végrehajtja a megfelelő gesztust.

Az előre meghatározott jelek felismerésére a ShiftTree által épített modellek felhasználhatóak. A modellek elkészítéséhez természetesen szükség van a felismerendő jelekre, mint idősorokra, de ezeket a kezdeti adatgyűjtési fázisban fel tudjuk venni. Szükség van az alapjelre is, ami a streamnek az a szakasza, ahol egyik felismerendő jel sem fordul elő. Emellett szükség van egy olyan kiterjesztésre, egy stream-kezelőre, ami az idősorok alapján épített ShiftTree modellt képes adatfolyamokon alkalmazni.

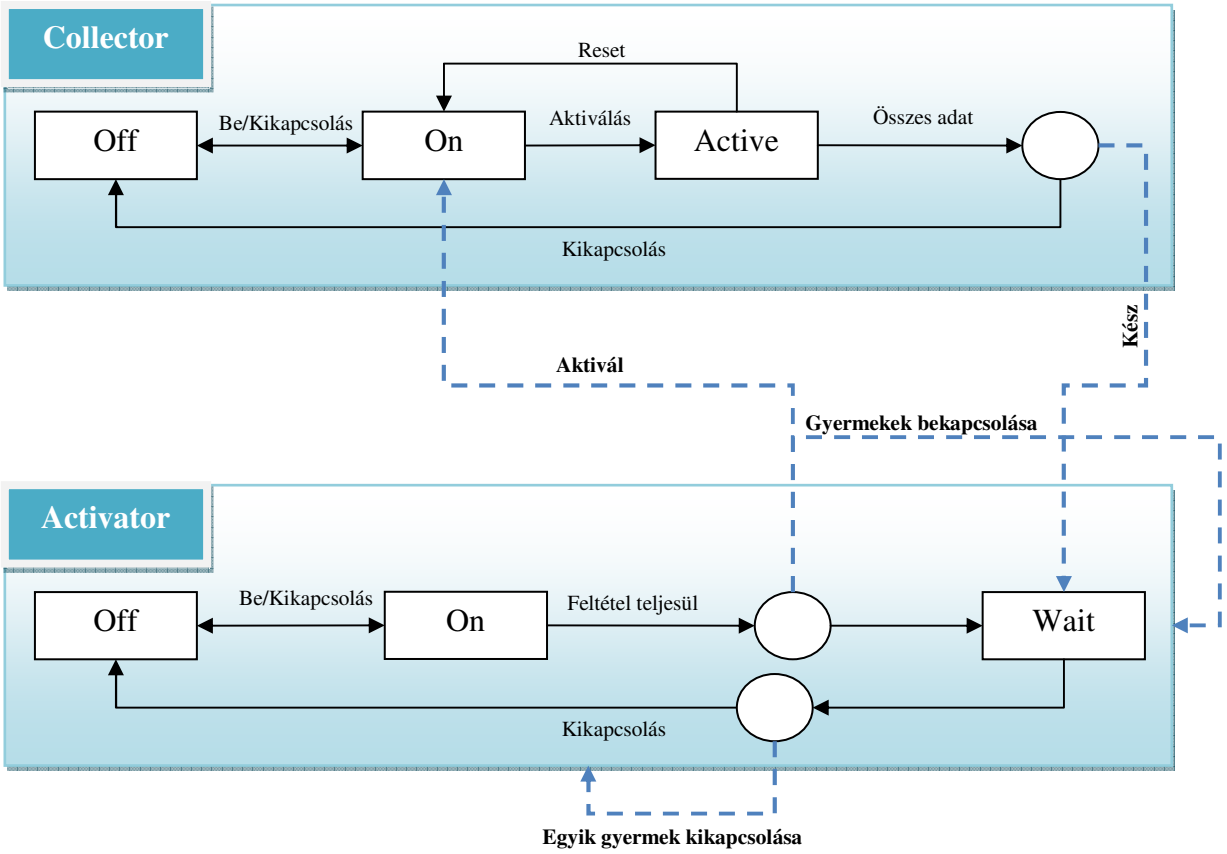
8.1.1. Csúszóablakos stream-kezelő

A stream-kezelő alapötlete, hogy az osztályozáshoz szükséges számítási műveleteket ne egyben végezzük el, hanem kenjük el. Elméletben megtehetnénk persze azt, hogy egy bufferbe gyűjtjük a streamből érkező adatokat, majd ha elég összegyűjt, akkor lefuttatjuk az osztályozást. De mivel nem tudjuk, hogy az modellünk által leírt idősorok a streamben hol kezdődnek, ezért a buffer megtelése után minden egyes új beérkezett adat esetén le kéne futtatni a címkéző algoritmust. Bár maga az osztályozás gyors, ha a bejövő adatok gyorsan érkeznek, akkor könnyen előfordulhat, hogy a processzor számítási kapacitása nem elég a felismerő futtatásához.

Ha egy csúszóablakos rendszert használunk, akkor ez nem okozhat problémát. Az alpötlet, hogy "kenjük" el az egyébként burstösen érkező számítási igényeket. Mindig amikor érkezik a következő adat, végezzük el azokat az apró műveleteket, amik előállítják a fa csomópontjaiban a döntéshez szükséges dinamikus attribútumot. Majd amikor elérkezik az a pillanat, amikor ezt az attribútumot vizsgálni kell (a szem a megfelelő pozícióba ért a

képzeletbeli idősoron), akkor vesszük az így előállt értéket, és összehasonlítjuk a modell adott csomópontjában lévő vágási értékkel.

Az eredeti algoritmus mintájára két operátorcsaládot érdemes itt is használni. Az első család a Collector-ok családja, amik az adatgyűjtésért felelősek. Ha egy Collector be van kapcsolva, akkor a beérkező adatot feldolgozza, úgy, hogy a benne lévő érték megegyezzen a modell csomópontjában vizsgált dinamikus attribútum értékével, vagy a lehető legkevesebb munkát kelljen elvégezni ahhoz, hogy ez az érték előálljon. A másik operátorcsalád az Activator-ok családja. Ezek az operátorok tartalmaznak egy Collectort, és figyelik a beérkező értékeket, és ha megadott feltételek teljesülnek, akkor aktiválja a hozzá rendelt Collectort, ami ha elég információval rendelkezik, akkor visszaadja az attribútum értéket, ha nem, akkor pedig elkezdi az adatgyűjtés utolsó fázisát, amiben előállítja az értéket (pl. környezet átlagolásakor kell előretekintés is). Ezen kívül az aktiválás pillanatában az Activator szól a fában egy szinttel lejjebb lévő Activator-oknak, hogy mostantól ők figyeljék, hogy az ő feltételük teljesül-e. Miután a Collector előáll az értékkel, a megfelelő gyermek Activator leáll. A 15. ábra szemlélteti az operátorok működését.



15. ábra: Activatorok és Collectorok működésének vázlata

Az operátorok be- és kikapcsolt állapotból is indulhatnak. Az ábrán látható körök azt jelzik, hogy az állapotváltás közben még végrehajt néhány dolgot az operátor. Ezeket a műveleteket a szaggatott nyilak is jelzik.

Egy Activator folyamatosan figyeli az adatfolyamot, így nem kell törődnünk azzal, hogy hol kezdődik a felismerendő jel a folyamban. Pl. egy Activator, ami az ESONextMax(1) operátornak felel meg, minden lokális maximumnál utasítja a megfelelő Collector-t, hogy adja vissza az attribútum értékét. Látható, hogy így egyszerre a legrosszabb esetben is annyi adatgyűjtő operátor (Collector) aktív, amennyi a fa csomópontjainak a száma. Mivel ez a szám jó modelleknél alacsony, és az adatgyűjtők csak néhány műveletet végeznek el minden

egyes lépésben, ezért a stream-kezelő folyamatosan alacsony számítási kapacitást igényel, nem pedig egyszerre sokat, és nem kell specifikálni számára azt, hogy hol kezdődik a felismerendő jel.

8.1.2. Nyitott kérdések

A ShiftTree modellek alkalmazása a jelfelismerési problémákra érdekes, ugyanakkor korántsem lezárt. Az egyik nyitott kérdés, hogy hogyan lehet a leghatékonyabban megkülönböztetni az egyes felismerendő jeleket az alapjeltől. Valószínűleg különböző szavazási struktúrákkal, különböző osztályokat elkülönítő modellekkel és az osztályozás konfidenciájával érdemes kísérletezni.

Egy fokkal nehezebb kérdés az, hogy hogyan lehet a jel hosszát érzékelni. Pl. a BCI esetében a tesztalanyunk hány másodpercig gondolt az email klienst elindító borítékra. A modelljeink csak azt mondják meg (jó esetben), hogy a borítékra gondolt-e. A jel végének megállapítása több szempontból is fontos: egyrészt ha túl korán indítjuk újra a felismerést, és még beleesünk az adott jel tartományába, akkor lehet, hogy egy parancsot kétszer ismerünk fel. Másrészt bizonyos parancsoknál (pl. scrollozás) fontos a jel tényleges hossza.

Egy harmadik nyitott kérdés az újratanulás. Egy ilyen rendszerben fontos az, hogy ha valamit rosszul ismert fel, akkor a felhasználó visszajelzése alapján módosítani tudjuk a modellt. Ugyanakkor tisztázandó, hogy milyen gyakran érdemes előről újraépíteni az egész modellt, és mikor elég az on-line tanulást alkalmazni. Illetve tisztázni kell, hogy ha a felhasználó nem jelzett vissza, akkor az mennyire vehető figyelembe, mint a modellünk pozitív megerősítése.

8.2. Címkézett gráfok és egyéb struktúrák osztályozása

A ShiftTree alapelve, miszerint egy adott struktúrán mozgunk és bizonyos pontokon előállítunk valamilyen tulajdonságok alapján dinamikus attribútumokat, kellően általános. Éppen ezért bármilyen olyan struktúrán használható, ahol képesek vagyunk értelmes operátorokat előállítani.

Az egyik ilyen alkalmazási terület lehet a címkézett gráfok osztályozása. A feladat lényege az, hogy van több gráfunk, amik valamilyen módon osztályokba vannak sorolva. Az élekhez és a csomópontokhoz is lehetnek értékek hozzárendelve, de az is lehet, hogy csak maga a struktúra adott. A feladatunk az, hogy ha kapunk egy új gráfot, akkor azt a jó osztályba soroljuk be. Látszik, hogy a probléma teljesen analóg az idősor osztályozással, csak itt más az adatok struktúrája. A gráfokon is nagyon jól el lehet különíteni az ESO-k és a CBO-k feladatát, így az algoritmus könnyen alkalmazható.

Nyitott kérdés viszont az, hogy mely pontból érdemes indulni. Az idősoroknál adja magát, hogy szem az elején az idősor elejére mutasson. Az általános gráfoknál viszont nincs egy ilyen kitüntetett csomópont, amit a kiinduláshoz használhatnánk, és minden gráf esetén hasonló a jelentése. Szintén érdekes kérdés, hogy a csomópontokat és az éleket hogyan érdemes megkülönböztetni egymástól, ha egyáltalán megkülönböztetjük őket (pl. egy operátor mozgathatja-e a szemet egy élre, vagy csak egy csomópontra).

Az alap ShiftTree által megoldott idősor-osztályozási problémához a legközelebb a többdimenziós idősorok osztályozása áll. Természetesen ehhez a problémához is új operátorok definiálása szükséges, mivel a mostaniak az egydimenziós világban dolgoznak.

Tehát a ShiftTree mögötti el sok irányba kiterjeszthető, és egy érdekes kutatási terület lehet az érdeklődők számára.

9. Összefoglalás

A 2. fejezetben bemutatott ShiftTree algoritmus egy modell alapú megoldás egy olyan területen (idősor-osztályozás), ahol a példány alapú módszerek túlsúlya jelentős. A modell alapú megoldásoknak számos előnye (és néhány hátránya) van. Az alap ShiftTree, amellett, hogy pontossága az elterjedt szomszédosság alapú módszerekével összevethető, még számos egyéb előnyös tulajdonsággal rendelkezik, mint pl. a probléma függetlenség, a szakértői tudás beépíthetősége, vagy éppen a modellek értelmezhetősége. Ezen tulajdonságok többsége a 7. fejezetben bemutatott példán is jól látszanak.

A 2008-as ShiftTree ugyanakkor még jelentősen tovább javítható. Az algoritmus sebessége két egyszerű megfontolást figyelembe véve úgy javítható több, mint 20%-kal, hogy közben nem veszítünk a pontosságából. Ráadásul ez a javítás előnyösen hat az algoritmus skálázódására is. (4.1. alfejezet)

Új, ugyanakkor nem bonyolult operátorok bevezetésével (3. fejezet), a módszer pontossága jelentősen növelhető, így már az Euklideszi távolságot használó szomszéd módszernél egyértelműen jobb eredmények érhetőek el, és a nagyobb adatsorokon, ahol elvárható, hogy a modell alapú módszerek működjenek, a DTW-t használó módszereknél is pontosabbá válik. Különféle, heurisztikán alapuló, tanítási módok (4.2. alfejezet) bevezetésével elérhető, hogy a modell egy kicsit pontosabbá, és sokkal robosztusabbá váljon, miközben a futási idő csak minimálisan emelkedik. (4.4. alfejezet). Érdekes módon a nyesési eljárások (4.3. alfejezet), legyenek akármilyenek, csak negatív hatásuk van a modell pontosságára. Ez egyben azt is jelenti, hogy a vizsgált adatokon nem jellemző az, hogy a ShiftTree túltanulna.

Modellek kombinációjával általában jelentősen jobb eredményeket lehet elérni, mint egy modellel. Nincs ez másként a ShiftTree esetében sem: az 5. fejezetben megvizsgált két forest eljárás - egy boosting eljárás és egy saját fejlesztésű, keresztvalidáción alapuló kombinálás - jelentősen javít az eredményeken. Ráadásul a két módszer kiegészíti egymást, más problémákon teljesítenek jól, így minden esetben tudunk megfelelő módszert választani. Így a módszer pontossága már nem marad el egy idősor-osztályozási versenyen kidolgozott más módszerek pontosságától, és bár a pontozás miatt nem kerül az első háromba, összesített pontosság alapján sokkal pontosabb, mint a többi, a versenyen szereplő megoldás.

Az konfidenciák bevezetésével (6. fejezet) már nem csak pontosságot használhatjuk a modellünk kiértékelésére, hanem a gyakran használt AUC-ot is. A ShiftTree ezen a téren sem szerepel rosszul, az eredmények eléggé meggyőzőek. Az útvonal konfidenciák koncepciójával lehetővé válik egy olyan dinamikus nyesés, ami egy kicsit javítja a modellt, ráadásul megteremti a lehetőséget az on-line tanításra. Ez utóbbival a módszer képessé válik arra, hogy a modellezési eljárás megismétlése nélkül, hatékonyan reagáljon arra, hogy idővel az adatok tulajdonságai változnak.

Mivel az algoritmus mögött álló elv kellően általános, egy algoritmus helyett igazából egy algoritmus családról beszélhetünk, ami többféle struktúrán is alkalmazható. Az idősor-osztályozásnál maradva azt is láthatjuk, hogy a ShiftTree modellek minimális számításigénnyel alkalmasak lehetnek a stream-mining egyik alapfeladatának, az adott típusú jelek felismerésének, megoldására (8. fejezet).

Összességében tehát elmondható, hogy a ShiftTree és a hozzá kapcsolódó fejlesztések az idősorok osztályozásának területén egy érdekes, és hatékony modell alapú módszert eredményeztek. A kiterjesztésekkel (stream-mining, változó adatok kezelése, forest módszerek), a módszer viszont már több területen átível, és megvan benne a potenciál, hogy a jövőben egy hasznos algoritmuscsaládként a legtöbb félig-strukturált, strukturált adat osztályozására alkalmassá váljon.

Függelék I. A felhasznált irodalom jegyzéke

[1] **Zoltán Prekopcsák:** Accelerometer Based Real-Time Gesture Recognition. *In POSTER 2008: Proceedings of the 12th International Student Conference on Electrical Engineering*. Prague, Czech Republic, May 2008.

[2] **Eamonn Keogh, Chotirat Ann Ratanamahatana:** Exact indexing of dynamic time warping. *In Knowledge and Information Systems (2005) 7: 358–386.*

[3] **Lawrence R. Rabiner:** A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE 77 (2): 257–286.* February 1989.

[4] **Eamonn Keogh, Xiaopeng Xi, Li Wei, Chotirat Ann Ratanamahatana:** The UCR Time Series Classification/Clustering Homepage (2006). URL: http://www.cs.ucr.edu/~eamonn/time_series_data/

[5] **Mahmoud Abou-Nasr, Lee Feldkamp:** Ford Classification Challange (2008). URL: http://home.comcast.net/~nn_classification/

[6] **Eamonn Keogh, Christian Shelton, Fabian Moerchen:** Workshop and Challenge on Time Series Classification at SIGKDD'07 (2007). URL: http://www.cs.ucr.edu/~eamonn/SIGKDD2007TimeSeries.html

[7] **The UCI KDD Archive**: Japanese Vowels (2008). URL: http://kdd.ics.uci.edu/databases/JapaneseVowels/JapaneseVowels.html

[8] **J. Fogarty, R. Baker, S. Hudson**: Case studies in the use of ROC curve analysis for sensor-based estimates in human computer interaction. *ACM International Conference Proceeding Series, Proceedings of Graphics Interface 2005.* Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society.

[9] **Hidasi Balázs**: Újfajta, automatikus, döntési fa alapú adatbányászati módszer idősorok osztályozására. (*BSc szakdolgozat, 2008. december*).

[10] **Breiman, Leo; Friedman, J. H., Olshen, R. A., & Stone, C. J.**: Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software (1984)

[11] **Ellis Horowitz, Sartaj Sahni**: Computing Partitions with Applications to the Knapsack Problem. (*JACM)*, Volume 21, Issue 2, 277-292, April 1974

[12] **J. R. Quinlan**: Induction of Decision Trees. *In Readings in Machine Learning.* (1990)

[13] **John Mingers:** An Empirical Comparison of Pruning Methods for Decision Tree Induction. *In Machine Learning, (1989) 2: 227-243.*

[14] **Yoav Freund, Robert E. Schapire:** A Short Introduction to Boosting. *In Journal of Japanese Society for Artificial Intelligence.* (1999)

[15] **Robert E. Schapire: The** Strength of Weak Learnability. (1990)

[16] **Yoav Freund, Robert E. Schapire:** A decision-theoretic generalization of on-line learning and an application to boosting. *In Journal of Computer and System Sciences 55(1):119–139.* August 1997.

[17] **Intelligent Data Analysis Group of Fraunhofer FIRST**: BCI Competition II. URL: http://www.bbci.de/competition/ii

[18] **Intelligent Data Analysis Group of Fraunhofer FIRST**: BCI Competition II, Data Set Ia: Self regulation of SCPs. URL: http://www.bbci.de/competition/ii/tuebingen_desc_i.html

Függelék II. Rövidítések jegyzéke

- AUC:
  - Area Uncer Curve
  - A ROC görbe alatti terület. Osztályozók minőségét jellemző mértékeknek.
- CBO:
  - ConditionBuilder Operator
  - A ShiftTree algoritmusban használt egyik operátortípus. A szem által mutatott értéket, a pozíciót, a környezetet, stb. felhasználva kiszámít egy értéket.
- CV:
  - Calculated Value
  - A CBO-k által számított érték.
- DTW:
  - Dynamic Time Wrapping
  - Idősorok közti távolság mérték, ami nem érzékeny pl. az időbeli eltolásra.
- ESO:
  - EyeShifterOperator
  - A ShiftTree algoritmusban használt egyik operátortípus. A szemet mozgatja az időtengely mentén egy meghatározott pozícióba.
- MM:
  - Multiple Model, Multiple Modelling.
  - A ShiftTree egyik tanítási módja, amelyben egy csomópontból több részfa is kiindulhat. Magát a modellt is szokás MM-mel rövidíteni.
- MM+, MM++, MM3+:
  - Olyan MM variánsok, amik valamilyen heurisztikát is használnak a csomópontokban a vágások kiválasztásakor.
- ROC:
  - Reciever Operator Characteristic
  - A hamis negatív arány és az érzékenység által meghatározott tér. Az egyes osztályozók teljesítménye elhelyezhető ebben a térben.
- SAMME:
  - Stagewise Additive Modeling using a Multi-class Exponential loss function
  - Egy boosting algoritmus, speciálisan a többosztályos esetre kifejlesztve.
- SM:
  - Simple Model, Simple Modelling
  - A ShiftTree alap tanítási módja: egy csomópontból egy részfa indulhat.
- SM+, SM++, SM3+:
  - Olyan SM variánsok, amik valamilyen heurisztikát is használnak a csomópontokban a vágások kiválasztásakor.
- TV:
  - Threshold Value
  - A ShiftTree egy csomópontjában szereplő referencia érték,
- VVO
  - Virtual Variable Operator
  - Egy új operátor a ShiftTree algoritmusban, ami virtuális (származtatott) változók létrehozásáért felelős.
- XV:
  - X-Validation (forest method)
  - Olyan forest eljárás, ami többféleképpen szétválasztott tanító és validációs halmazokon (keresztvalidáción) alapul.

Függelék III. Ábrák jegyzéke

Függelék IV. Táblázatok jegyzéke

Függelék V. Algoritmusok jegyzéke

Függelék VI. Tesztelési konfigurációk

Alap operátorkészlet:

ESONextMax(1) ESONextMin(1) ESONext(1) ESONext(5) ESONext(10) ESONext(25)
ESONext(50) ESONext(100) ESONext(200) ESONext(400) ESOMax(global) ESOMin(global)
ESOPrev(1) ESOPrev(5) ESOPrev(10) ESOPrev(25) ESOPrev(50) ESOPrev(100)
ESOPrev(200) ESOPrev(400) ESOPrevMax(1) ESOPrevMin(1)

CBOSimple CBONormal(0,1,3) CBOExp(1,3) CBONormal(1,0.5,3) CBOExp(0.5,8)
CBONormal(0.5,4,10) CBOExp(2,3) CBOAvg(5)

Bővített operátorkészlet:

ESONext(1) ESOPrev(1) ESONext(11%) ESOPrev(11%) ESONext(22%) ESOPrev(22%)
ESONext(33%) ESOPrev(33%) ESONext(44%) ESOPrev(44%) ESONext(55%) ESOPrev(55%)
ESONext(66%) ESOPrev(66%) ESONext(77%) ESOPrev(77%) ESONext(88%) ESOPrev(88%)
ESONext(99%) ESOPrev(99%) ESONextMax(1) ESONextMin(1) ESONextMax(2) ESONextMin(2)
ESONextMax(5) ESONextMin(5) ESONextMax(10) ESONextMin(10) ESONextMax(17) ESONextMin(17)
ESOPrevMax(1) ESOPrevMin(1) ESOPrevMax(2) ESOPrevMin(2) ESOPrevMax(5) ESOPrevMin(5)
ESOPrevMax(10) ESOPrevMin(10) ESOPrevMax(17) ESOPrevMin(17) ESOClosestMax ESOClosestMin
ESOGreaterMax(norm) ESOGreaterMin(norm) ESOLesserMax(norm) ESOLesserMin(norm)
ESOMax(global) ESOMin(global) ESOMax(sofar) ESOMin(sofar) ESONext(10) ESONext(25)
ESONext(50) ESONext(100) ESONext(150) ESONext(200) ESONext(300) ESONext(400)
ESOPrev(10) ESOPrev(25) ESOPrev(50) ESOPrev(100) ESOPrev(150) ESOPrev(200)
ESOPrev(300) ESOPrev(400) ESOMaxInNextInterval(20) ESOMaxInNextInterval(45)
ESOMaxInNextInterval(90) ESOMaxInNextInterval(180) ESOMinInNextInterval(20)
ESOMinInNextInterval(45) ESOMinInNextInterval(90) ESOMinInNextInterval(180) ESOMaxInPrevInterval(20)
ESOMaxInPrevInterval(45) ESOMaxInPrevInterval(90) ESOMaxInPrevInterval(180) ESOMinInPrevInterval(20)
ESOMinInPrevInterval(45) ESOMinInPrevInterval(90) ESOMinInPrevInterval(180)
ComplexESO(ESONext(90),ESOClosestMax) ComplexESO(ESONext(90),ESOClosestMin)
ComplexESO(ESOPrev(90),ESOClosestMax) ComplexESO(ESOPrev(90),ESOClosestMin)
ComplexESO(ESONext(180),ESOClosestMax) ComplexESO(ESONext(180),ESOClosestMin)
ComplexESO(ESOPrev(180),ESOClosestMax) ComplexESO(ESOPrev(180),ESOClosestMin)
ComplexESO(ESONext(90),ESOGreaterMax(norm)) ComplexESO(ESOPrev(90),ESOGreaterMax(norm))
ComplexESO(ESONext(90),ESOLesserMin(norm)) ComplexESO(ESOPrev(90),ESOLesserMin(norm))
ComplexESO(ESONext(180),ESOGreaterMax(norm)) ComplexESO(ESOPrev(180),ESOGreaterMax(norm))
ComplexESO(ESONext(180),ESOLesserMin(norm)) ComplexESO(ESOPrev(180),ESOLesserMin(norm))
ComplexESO(ESONext(90),ESOMaxInNextInterval(90)) ComplexESO(ESOPrev(90),ESOMaxInPrevInterval(90))
ComplexESO(ESONext(90),ESOMaxInPrevInterval(90)) ComplexESO(ESOPrev(90),ESOMaxInNextInterval(90))
ComplexESO(ESONext(90),ESOMinInNextInterval(90)) ComplexESO(ESOPrev(90),ESOMinInPrevInterval(90))
ComplexESO(ESONext(90),ESOMinInPrevInterval(90)) ComplexESO(ESOPrev(90),ESOMinInNextInterval(90))
ComplexESO(ESOMax(global),ESOMaxInNextInterval(90)) ComplexESO(ESOMax(global),ESOMaxInPrevInterval(90))
ComplexESO(ESOMin(global),ESOMinInPrevInterval(90)) ComplexESO(ESOMin(global),ESOMinInNextInterval(90))
ComplexESO(ESONext(45),ESOClosestMax) ComplexESO(ESONext(45),ESOClosestMin)
ComplexESO(ESOPrev(45),ESOClosestMax) ComplexESO(ESOPrev(45),ESOClosestMin)
ComplexESO(ESONext(45),ESOGreaterMax(norm)) ComplexESO(ESOPrev(45),ESOGreaterMax(norm))
ComplexESO(ESONext(45),ESOLesserMin(norm)) ComplexESO(ESOPrev(45),ESOLesserMin(norm))
ESOBegin ESOEnd ESOStay ESOClosestLocal ESOGreaterDistLocal(norm) ESOGreaterMax(abs)
ESOGreaterMin(abs) ESOGreaterDistLocal(abs) ESOLesserDistLocal(norm) ESOLesserMax(abs)
ESOLesserMin(abs) ESOLesserDistLocal(abs)

CBOSimple CBONormal(0,1,3) CBONormal(1,0.5,3) CBONormal(0.5,4,10)
CBONormal(0,4,9) CBONormal(0,4,5) CBONormal(0,8,10) CBONormal(0,16,20)
CBOExp(1,3) CBOExp(0.5,8) CBOExp(2,3) CBOExp(6,5) CBOExp(6,10) CBOExp(12,20)
CBOAvg(5) CBOAvg(10) CBOAvg(20) CBOAvg(43) CBOAvg(60) CBOAvg(86)
CBOLinear(5) CBOLinear(10) CBOLinear(20) CBOLinear(43) CBOLinear(60) CBOLinear(86)
CBODeltaT(abs) CBODeltaT(delta) CBOTimeSensitive(abs) CBOTimeSensitive(delta) CBOAverage(sofar)
CBOAverage(delta) CBOVariance(sofar) CBOVariance(delta) CBOMaxAvg(sofar)
CBOMaxAvg(delta) CBOMinAvg(sofar) CBOMinAvg(delta) CBOMaxVar(sofar)
CBOMaxVar(delta) CBOMinVar(sofar) CBOMinVar(delta) CBOMaxCount(sofar)
CBOMaxCount(delta) CBOMinCount(sofar) CBOMinCount(delta) CBOMedian(5,5)
CBOMedian(10,10)

BCI operátorkészlet:

ESONextMax(1) ESONextMin(1) ESONextMax(2) ESONextMin(2) ESONextMax(5) ESONextMin(5)
ESONextMax(10) ESONextMin(10) ESOPrevMax(1) ESOPrevMin(1) ESOPrevMax(2) ESOPrevMin(2)
ESOPrevMax(5) ESOPrevMin(5) ESOPrevMax(10) ESOPrevMin(10) ESOClosestMax ESOClosestMin
ESOGreaterMax(abs) ESOGreaterMin(abs) ESOLesserMax(abs) ESOLesserMin(abs)
ESOGreaterMax(norm) ESOGreaterMin(norm) ESOLesserMax(norm) ESOLesserMin(norm)
ESOMax(sofar) ESOMin(sofar) ESONext(1) ESONext(12) ESONext(24) ESONext(48)
ESONext(96) ESONext(192) ESOPrev(1) ESOPrev(12) ESOPrev(24) ESOPrev(48)
ESOPrev(96) ESOPrev(192) Complex(ESONext(24),ESOClosestMax)
Complex(ESONext(24),ESOClosestMin) Complex(ESOPrev(24),ESOClosestMax)
Complex(ESOPrev(104),ESOClosestMin) Complex(ESONext(104),ESOClosestMax)
Complex(ESONext(104),ESOClosestMin) Complex(ESOPrev(104),ESOClosestMax)
Complex(ESOPrev(104),ESOClosestMin)

CBOSimple CBONormal(0,1,3) CBONormal(0,1,6) CBONormal(0,4,6)
CBONormal(0,4,12) CBONormal(0,4,24) CBOExp(1,3) CBOExp(1,6) CBOExp(2,3)
CBOExp(2,6) CBOExp(2,12) CBOExp(4,24) CBOAvg(6) CBOAvg(12) CBOAvg(24)
CBOAvg(36) CBOAvg(48) CBOAvg(72) CBOLinear(6) CBOLinear(12) CBOLinear(24)
CBOLinear(36) CBOLinear(48) CBOLinear(72) CBODeltaT(abs) CBODeltaT(delta) CBOTimeSensitive(abs)
CBOTimeSensitive(delta) CBOAverage(sofar) CBOAverage(delta) CBOVariance(sofar)
CBOVariance(delta) CBOMaxAvg(sofar) CBOMaxAvg(delta) CBOMinAvg(sofar)
CBOMinAvg(delta) CBOMaxVar(sofar) CBOMaxVar(delta) CBOMinVar(sofar)
CBOMinVar(delta) CBOMaxCount(sofar) CBOMaxCount(delta) CBOMinCount(sofar)
CBOMinCount(delta) CBOMedian(6,6) CBOMedian(12,12) CBOMedian(24,24)

VVOAvg(0,1) VVOAvg(2,3) VVOAvg(4,5) VVOSub(norm,0,1) VVOSub(norm,2,3)
VVOSub(norm,4,5) VVOSub(abs,0,1) VVOSub(abs,2,3) VVOSub(abs,4,5)