SZAKDOLGOZAT

ÚJFAJTA, AUTOMATIKUS, DÖNTÉSI FA ALAPÚ ADATBÁNYÁSZATI MÓDSZER IDŐSOROK OSZTÁLYOZÁSÁRA

Hidasi Balázs
BHIDASI@T-ONLINE.HU

Konzulens:

Gáspár-Papanek Csaba
ügyvivő szakértő, Távközlési és Médiainformatikai Tanszék
GASPAR@TMIT.BME.HU

BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
TÁVKÖZLÉSI ÉS MÉDIAINFORMATIKAI TANSZÉK

Budapest, 2008. december

KIVONAT

Az osztályozás az adatbányászat egyik fő területe. A fix hosszúságú adatrekordok osztályozására már régóta léteznek letisztult algoritmusok. Az idősorok osztályozása viszont napjaink egyik fontos kutatási területe az adatbányászatban. A hagyományos módszerek átiratai általában hosszú adatelőkészítési fázist igényelnek, tehát nem eléggé automatikusak. Sok módszer addig transzformálja az adatokat, amíg az egyik legfontosabb dolog, az időbeliség el nem veszik. A tématerület specifikus osztályozók a területhez kapcsolódó idősorokat kiemelkedő pontossággal osztályozzák, de más területen alkalmazva pontatlanok, azaz nem eléggé általánosak. Más módszerek nem eléggé pontosak, és a megoldások nagy része nem ad magyarázatot arra, hogy miért úgy osztályozott, ahogy. A megmagyarázhatóság a gyakorlati alkalmazások körében viszont sokszor fontosabb követelmény, mint maga a pontosság.

Az általam kifejlesztett ShiftTree módszer az említett három követelményeknek próbál egyszerre megfelelni, tehát egyszerre próbál automatikus, pontos és magyarázó lenni. Az algoritmushoz a bináris döntési fákat használtam alapként, de jelentősen átalakítottam a csomópontok szerkezetét, azokat három fő funkcionális egységre bontottam. Mindegyik funkcionális egységhez adott típusú egyszerű operátorok tartoznak. A funkcionális egységek az operátorok egyszerű lépéseit ismételve egy komplex osztályozót hoznak létre. A részegységek úgy lettek megalkotva, hogy könnyen bővíthetőek legyenek új operátorokkal, ami által speciális adatsorokhoz a tématerület szakértője által definiált speciális operátorok is könnyen beilleszthetővé válnak. Az említettek közül az automatizmus az, amit a kidolgozás során különösen szem előtt tartottam. A módszer emiatt rugalmasan kezeli a különböző hosszú idősorokat, akár egy osztályozási feladaton belül is, ami által az adatelőkészítési fázis jelentősen lerövidül, és életszerű adatok feldolgozására is alkalmassá válik az eljárás. Szintén az automatizmusnak köszönhető, hogy kezeli egyaránt az egy- és a többattribútumos idősorokat is.

Az elméleti kidolgozás után implementáltam a módszert, hogy megnézhessem, hogy hogyan teljesít a gyakorlatban. Az elkészült programot nemzetközi benchmark adatbázison (20 különböző adatsor) és egyéb a valós életből vett idősorokon (gyorsulásmérő adatai, digitalizált hangok, gyártósorokhoz tartozó mérések, stb.) futtattam – többféle beállítás mellett – és pontosság szempontjából összehasonlítottam több, jelenleg népszerű idősor-osztályozóval. A módszerem számos esetben felülmúlta a többi módszer pontosságát. Emellett nagyszámú tesztet végeztem annak kiderítésére, hogy a tanítópontok számának növekedése és az idősorok egyéb tulajdonságai hogyan hatnak a futási időre, és az osztályozás pontosságára. Az eredményeket összehasonlítottam az elméleti számításaimmal, az eltérések okait felkutattam és megmagyaráztam. Mivel a módszer egyik célja az volt, hogy magyarázó legyen, megvizsgáltam az algoritmus által néhány adatsorhoz épített modelleket, hogy valóban könnyen értelmezni tudom-e őket. Ezek az elemzések igazolták, hogy az eljárás jól értelmezhető, helyes modelleket generál.

Összességében az látszik, hogy a módszer magas potenciállal rendelkezik az idősorok osztályozása terén, köszönhetően annak, hogy automatikus, jól magyaráz, kellően pontos, megfelelően gyors és könnyen bővíthető.

Kulcsszavak: adatbányászat, idősor-osztályozás, döntési fa

ABSTRACT

Classification is one of the main fields of data mining. Classification algorithms for data records have been existing for a long time. But time series classification is one of the most important field of research nowadays in data mining. The variants of the classic methods require a great amount of human work in the data preparation phase. Most of the classic algorithms don't care about the sequence of the elements, which is great loss of information by time series classification. There are field-specific time series classificators which operate on a high level of efficiency in their fields. However they are inefficient on other fields because they are too specific. Other methods are inaccurate and most of the algorithms fail at giving an explanation of their model, which is rather important in practise.

ShiftTree is a decision tree based time series classification algorithm that is developed by me. I created this method to fit three requirements. It should be automated and general: it should require less amount of human work than other methods and it should work on many types of time series. The second requirement is accuracy: it should do the classification task with high accuracy. The last one is explainability: the classification model should stand for itself; it should give some intuition about the indexing.

I used binary decision trees as the base of the ShiftTree and I modified the nodes of the tree to have three subsystems in all nodes. Each subsystem has its own role and contains operators of one type. By repeatedly using the simple operators, the three subsystems become a complex classificator. Operators can be easily added to the subsystems in order to let the expert of the application field define and add some that might be useful in the current classification task. The algorithm is automated so it can handle time series of any length, with any number of classes. It can also handle time series with more than one attributes (multidimensional time series).

I also implemented the algorithm to see how it performs. I tested the program on an international benchmark database which consists of 20 datasets from 20 fields of application. I also tested it on other data like digitalized sounds, data from the built in accelerometers of mobile phones, etc. I compared the results on accuracy with the accuracy of other algorithms. The ShiftTree is often better then the other algorithms I used.

I made large number of tests in order to find out how the size of the training set and other properties of the datasets influence the accuracy of the indexing and the time of building the model. I compared the results with my theoretical estimates. If there were differences, I found out the reason behind them and explained them.

One of the requirements was explainability. Therefore I analyzed some models to see if they can explain the indexing process. These analyses proved that the algorithm creates easily explainable models.

On the whole it seems that the ShiftTree has a great potential is time series classification, because it is automated, general, accurate, creates explainable models, the time of building the model is acceptable and new operators can be easily added to it.

Keywords: data mining, time series, classification, decision trees

TARTALOMJEGYZÉK

1. A probléma leírása

1.1. Alapfogalmak és értelmezésük

1.1.1. Az osztályozási feladat

Az osztályozás egyike az adatbányászat fő területeinek. A feladat az, hogy adott típusú adatrekordokat előre meghatározott kategóriákba, osztályokba soroljunk.

Az osztályozás általánosan két lépésben kerül megoldásra. Az első lépés a modellezés, ahol olyan adatrekordokat használunk, amelyekhez ismerjük az osztályváltozó értékét, ezek az úgynevezett tanítópontok. A modellt ezeken a rekordokon tanítjuk, az eredmény egy olyan összefüggés-rendszer lesz, ami a rekordok egyes attribútumainak értéke alapján megadja az osztályváltozó értékét. Az osztályozás második lépése a modell alkalmazása, azaz az ismeretlen osztályváltozójú rekordok osztályváltozójának megállapítása.

1.1.2. Idősorok

Az idősor [1] egy olyan függvény, ami egy vagy több változó értékét ábrázolja egy vagy több idő-szerű változó függvényében. Idő-szerű változó alatt azt értem, hogy a változó olyan jelentést hordoz magában, hogy ami ennek a változónak a T értékénél történt, az megelőzi valamilyen módon azt, ami a T+1 értéknél történt. Ez a változó általában az idő, de lehet mondjuk a tér valamelyik koordinátatengelye is. Ezzel az általános megfogalmazással az összes idősornak tekinthető adatot leírtam, a következőkben az egyes jelentősebb típusokat részletesebben jellemzem.

1.1.2.1. Klasszikus idősorok

A klasszikus vagy egyváltozós idősoroknak egy változójuk és egy időtengelyük van. Ilyen adat például egy tőzsdeindex értékeinek sorozata egy nap folyamán, vagy mondjuk a napi középhőmérsékletek éves sorozata. Az ilyen jellegű idősoroknál az idő-szerű változó szinte minden esetben maga az idő.

1.1.2.2. Többváltozós idősorok

A többváltozós vagy többattribútumos idősoroknak szintén egy időtengelyük van, viszont minden időponthoz több változóérték tartozik. A többattribútumos idősorok szétbonthatóak annyi egyváltozós idősorra, amennyi az attribútumaik száma, de a változók együttes értéke szintén információt hordoz, ezért a szétbontásuk nem ajánlott. Ilyen adat például egy gyorsulásmérő mérési eredménye, ami az X, Y és Z koordinátatengelyek irányába történő gyorsulás mértékét adja vissza minden századmásodpercben.

1.1.2.3. Többdimenziós idősorok

A többdimenziós idősoroknak egynél több időtengelye van. Itt is megkülönböztethetünk egy- és többváltozós idősorokat. Ebben az esetben az időtengelyek általában térbeli koordinátatengelyek, esetleg az idő is szerepelhet, mint az egyik koordináta tengely. Egy 2D-s idősor például egy szürkeárnyalatos kép, ahol a két időtengely az X és az Y koordinátatengely, a változó pedig az egyes pixelek világosság értékét adja. Egy 4D-s idősor például egy MRI felvétel, ahol a négy időtengely az X, Y, Z koordinátatengelyek és az idő, a változó pedig az adott pont színe az adott pillanatban.

1.2. Motiváció

Rengeteg olyan adat áll rendelkezésünkre, ahol fontos az adatok sorrendje, időbelisége. Azt is mondhatjuk, hogy az adatok nagy részét az idő függvényében tároljuk.

Az idősorok hatékony osztályozása szolgálhat diagnosztikai célokat: például EEG felvételeket sorolhatunk automatikusan „normális" és „gyanús" kategóriákba. Segíthet a hang alapú azonosításban: osztályozhatunk digitalizált hangfelvételeket az alapján, hogy ki mondta ki őket. De a fentebb említettek alapján ide tartozik a képek felismerése is, ami mostanában, az intelligens megfigyelőrendszerek miatt, ismét az érdeklődés középpontjába került.

1.3. Idősor osztályozó algoritmusok/módszerek

Az idősor-osztályozás területén már sok algoritmus született. Ebben a pontban az általános megközelítési módok ismertetése után röviden leírok néhány elterjedt, klasszikus osztályozási módszert, amiket vektorizált idősorok osztályozására is szoktak alkalmazni.

1.3.1. Általános megközelítési módok

Az idősorok osztályozását több irányból szokás megközelíteni, ezek: attribútumok kinyerése, vektorrá alakítás, új osztályozási algoritmusok speciálisan az idősorokra.

1.3.1.1. Attribútumok kinyerése

Az attribútumok kinyerésének az a lényege, hogy néhány, az idősorra jellemző statisztikai mennyiséget származtatunk az idősorainkból, és ezeket használjuk, mint az osztályozáshoz szükséges attribútumokat. Az osztályozáshoz a már jól bevált osztályozó módszereket alkalmazzuk. A módszer hátránya, hogy ezek az attribútumok sokszor nem jellemzik elég jól az idősorokat, és emiatt pontatlan eredményeket kaphatunk.

1.3.1.2. Vektorrá alakítás

A vektorrá alakítás módszere szintén a klasszikus osztályozókat használja. Attribútumoknak viszont az idősor változójának értékeit használja. De mivel a klasszikus osztályozók fix mennyiségű attribútumot képesek kezelni, ezért az idősorokon különböző transzformációkat hajtanak végre az osztályozás előtt, ami által egy adott hosszú vektort kapunk. Az eljárás kombinálható az előzővel, azaz a vektor értékeihez hozzávehetjük a statisztikai módszerekkel kinyert attribútumokat is.

1.3.1.3. Új algoritmusok

A harmadik módszer az, hogy új osztályozási algoritmusokat fejlesztenek ki speciálisan az idősorok számára. Ilyen módszerek lehetnek például a részgörbék alapján történő gyakori mintakeresési eljárások. Az ilyen algoritmusok kutatása még gyerekcipőben jár.

1.3.2. A Knn algoritmus

A Knn vagy más néven a K-legközelebbi szomszéd algoritmus [2] egy klasszikus osztályozó, amit fix hosszú vektorizált idősorok osztályozására is használnak. Úgy működik, hogy egy megadott távolságdefiníció alapján (ami általában az euklideszi távolság) az osztályozandó rekordhoz megkeresi a tanítóhalmazban a K legközelebbi tanítópontot és többségi szavazás alapján megállapítja az osztályváltozó értékét. Ez az egyik legegyszerűbb klasszikus osztályozó, aminek az alapváltozata is rendkívül érzékeny az osztályváltozótól független változókra és a változók értékkészletének nagyságára. Ezek a hiányosságok idősor-osztályozás esetén is fennállnak.

1.3.3. Döntési fa

A döntési fa [3] szintén egy elterjedt, klasszikus osztályozó, ami a klasszikus osztályozási feladatoknál általában jól teljesít. A módszer lényege, hogy egy csomópontban megvizsgáljuk, hogy a csomópontba került rekordokat hogyan vághatnánk a legjobban részekre. Ennek eldöntéséhez a rekordok attribútumaihoz feltételeket állít, majd egy beépített jóságértéket számít. Ennek az értéknek a kiszámítása algoritmus függő. A legjobb vágást a csomópont végrehajtja, és az így keletkezett ponthalmazokat eggyel lejjebbi szinten lévő csomópontoknak adja tovább. Az épített fa lehet bináris, illetve nem bináris, az algoritmustól függően. Több döntési fa alapú algoritmus létezik, a legismertebbek a C4.5, a C5.0, a CART és a CHAID. A módszer egyik legnagyobb előnye, hogy a felhasználó számára is könnyen értelmezhető szabályokat ad az osztályozás mikéntjére vonatkozóan. Idősorokra alkalmazva ez a magyarázó-erő általában elveszik.

1.3.4. Szupport vektor gépek (SVM)

Egy újabb klasszikus osztályozó eljárás. Az SVM [4] egy bináris osztályozó eljárás, azaz igen-nem jellegű osztályozásokra képes. Több osztály esetén ilyen osztályozások sorozatát alkalmazzuk. A módszer lényege, hogy egy sokdimenziós térben pontoknak tekinti a rekordokat, amiknek az egyes koordinátái a rekordok attribútumainak értékei. Ezt a sokdimenziós teret egy speciális képlettel magasabb dimenzióba transzformálja, és ott egy olyan hipersíkot keres, ami a tanítópontok két osztályát a legjobban szétválasztja. Emellett a két osztály pontjainak a vágó síktól lévő távolságát maximalizálja. Nagyon pontos módszer, viszont jelentős erőforrásokat igényel, ami a dimenziók (attribútumok) számának növekedésével jelentősen nő. Idősorok esetén azért alkalmazható nehezen, mert ha egy idősort egy vektornak tekintünk, akkor a vektor a legtöbb esetben több száz elemet tartalmaz, ami az SVM számára több száz attribútumot, és így dimenziót jelent.

1.3.5. A Random Forest eljárás

A Random Forest [5] egy újabb algoritmus, amit klasszikus osztályozónak fejlesztettek, de vektorizált idősorok osztályozására is alkalmazzák. A módszer lényege, hogy nem egy, hanem több döntési fát épít, méghozzá úgy, hogy minden csomópontban véletlenszerűen választ k darab olyan attribútumot, amit a vágáshoz használni fog. Ez a k szám jóval kisebb, mint az összes attribútum száma. Minden fához kettéosztja a tanítópontokat egy tanító- és egy teszthalmazra (visszatevéses módon választ ki N tanítópontot az N tanítópont közül, a maradék a teszthalmaz). Az előbbin tanítja a fát, az utóbbin számolja ki a hibát. Az osztályváltozót a fák és azok hibája alapján számítja ki az algoritmus.

1.3.6. Bayes-hálók

Független változók esetén a Bayes-tételből – a valószínűségeket relatív gyakoriságokkal közelítve – kiszámolható, hogy adott attribútum értékek mellett melyik osztálynak mennyi a valószínűsége. Arra az osztályra döntünk, aminek az adott értékek mellett a legnagyobb a valószínűsége. Ez viszont csak független attribútumok esetén lehetséges, amit általános esetben sem feltételezhetünk, idősorok esetében pedig végképp nem, hiszen függvényekről van szó, ahol az adott érték nagyban függ az előtte lévőktől. A Bayes-háló [6] egy irányított, körmentes gráf, ahol az egyes élek a függőségeket jelölik. Ezt a struktúrát előre meg lehet adni, ami jó abban az esetben, ha van erre vonatkozóan apriori tudásunk. Ha nincs, akkor a hálózat topológiáját automatikusan is ki lehet alakítani. A függőségek feltárása után, azok figyelembe vételével a valószínűségek már kiszámolhatóak.

1.3.7. Többrétegű neurális hálók (MPL)

A mesterséges neurális hálók [7] szintén bináris osztályozók, amik mesterséges neuronokat tartalmaznak. Minden neuronnak vannak bemenetei és van egy kimenete. A bemeneteken különböző súlyok vannak. Maga a neuron a súlyokkal felszorozva összegzi a bemenetein megjelenő értékeket, majd a kapott értéket egy szigmoid függvénnyel transzformálja, és ezt az értéket adja ki a kimenetén. A neuronok bemeneteire kerülhetnek az egyes attribútumok értékei, vagy más neuronok kimenetei, a neuronok kimenetei adhatják az osztályváltozó értékét vagy más neuronok bemeneteiként szolgálnak. A többrétegű modellben a neuronok rétegekbe szerveződnek úgy, hogy az i. rétegben lévő neuronok bemenetére csak az i-1. rétegben lévő neuronok kimenetei lehetnek rákötve (az utolsó réteg egy neuront tartalmaz, mert egy kimenetet várunk). A modell tanítása a súlyok automatikus – a tanítópontok alapján történő – változtatásával történik. A módszer egyike a leghatékonyabb klasszikus osztályozóknak, de egyben ez adja a legkevesebb magyarázatot a felépített modell működésére.

1.3.8. Logistic Model Tree (LMT)

Az LMT algoritmus [8] kombinálja a döntési fákat a logisztikus regresszióval. Minden csomópontban egy logisztikus modellt épít az oda került rekordok alapján, kiindulási súlyoknak és egyéb paramétereknek a szülő csomópont végleges modelljében szereplő értékeket használva. Az osztályozás során a fa feltételei mellett a csomópontokban lévő modellt is használja az osztályváltozó megállapításához.

1.4. Az idősorok osztályozásának problémái

A jelenlegi módszerek általában a fent is említett klasszikus osztályozási algoritmusokat használják, de az idősorok miatt az adatelőkészítési fázis rengeteg időt igényel, mivel a klasszikus osztályozók nem idősor jellegű adatokra lettek kitalálva. Ráadásul ez az előkészítési fázis idősor típusonként más, és más, ezért nagymennyiségű emberi munkát igényel. A klasszikus algoritmusok és az adatok összeegyeztethetőségének nehézsége onnan adódik, hogy az idősorok hossza nem feltétlenül azonos minden esetben. Sőt, az a jellemző, hogy a gyakorlati problémáknál a hosszak eltérnek. Például egy hang alapú azonosító rendszernél nem várhatjuk el, hogy az összes jogosult pontosan ugyanannyi idő alatt mondja ki a jelszót, ami alapján azonosítjuk a hangjukat. Szintén komoly probléma, hogy a klasszikus osztályozók nem kezelik az időbeliséget, tehát pont a probléma lényegét megtestesítő információ veszik el.

A modern módszerek hátránya, hogy sokszor csak egy-egy problématerületre koncentrálnak, azaz csak adott jellegű idősor(ok) osztályozására képesek csak hatékonyan. Bár a saját területükön belül kimagasló eredményeket érnek el, más területeken való hatékony alkalmazásuk nehéz vagy sokszor lehetetlen. Az általánosan működő modern módszerek meg általában nem eléggé pontosak. Ezen kívül a legtöbb módszernek gondot okoz, ha több vagy kevesebb változós idősorokat kap, mint amire ki lett fejlesztve.

Emellett szinte egyik módszer sem ad magyarázatot az osztályozás mikéntjére, ami azért lenne nagyon fontos, hogy az alkalmazó ellenőrizni tudja, hogy a modell olyan összefüggéseket tanult meg, amik általánosan is érvényesek és értelmesek. Ennek ellenőrzése nélkül a gyakorlati alkalmazók aligha fognak megbízni a felépített modellekben annyira, hogy a pénzüket/életüket rá merjék bízni.

2. A ShiftTree algoritmus

A ShiftTree egy teljesen új, döntési fa alapú algoritmus, amit idősorok osztályozására fejlesztettem ki. A kifejlesztéskor azt tartottam szem előtt, hogy az algoritmus lehetőleg minél általánosabb legyen, azaz bármilyen idősor-osztályozási feladattal jól boldoguljon. Ebben a fejezetben ismertetem a módszer elvi felépítését, majd részletesen írok a modellépítés és az osztályozás folyamatáról. A fejezet végén kitérek az egy- és többattribútumos változatok közti eltérések mibenlétére.

2.1. A ShiftTree felépítése, a részegységek feladata

A ShiftTree egy döntési fa alapú módszer. Az eredeti elképzeléshez képest csak a csomópontok szerkezetét változtattam meg, de azt jelentősen, hogy a módszer alkalmas legyen az idősorok osztályozására. Minden csomópont három modulból áll, ezek a szemtologató (eyeshifter), a feltételállító (conditionbuilder) és a döntő (decider). Az első két modul egy vagy több megfelelő operátort tartalmaz, az utóbbi mindig egy operátorral rendelkezik. Ebben a pontban a felhasznált döntési fa alapok rövid ismertetése után ismertetem az egyes részegységeket, és azok együttműködését.

2.1.1. A módszer alapjáról

A ShiftTree algoritmus alapját a bináris döntési fák szolgáltatják. Ezek olyan döntési fák, amelyekben minden csomópontnak 0 vagy 2 gyermeke lehet. A bináris döntési fák alapötlete, hogy minden csomópontba kerül valamennyi pont a tanítóhalmazból (a gyökér csomópontban ott van a teljes tanítóhalmaz), és a nem-levél csomópontok ezeket a pontokat osztják ketté egy általuk megállapított feltétel alapján. A feltételt úgy választják ki, hogy minden attribútumra megfogalmaznak minden lehetséges vágást, és megnézik, hogy így mi az a két halmaz, amit kaptak. Erre az eloszlásra egy jóságértéket számolnak egy adott függvény alapján, és a legjobb vágást alkalmazzák. A fa építése több esetben is véget érhet, ez szintén algoritmusfüggő.

Az osztályozás ezután annyiból áll, hogy az osztályozni kívánt adatot odaadjuk a gyökérnek, ami megvizsgálja, hogy a feltétel teljesül-e. Ha teljesül, akkor az egyik, ha nem, akkor a másik gyermekének adja tovább. Ez rekurzívan egészen addig folytatódik, amíg a pont el nem ért egy levelet. A levélben, a modellépítés során odakerült tanítópontok osztályváltozóinak értékei alapján, többségi szavazással megállapítjuk az osztályozandó rekord osztályváltozóját.

Ezek azok az alapok, amiket a döntési fákból alapul vettem. Emellett fontos megjegyezni, hogy egyelőre kizárólag idősorok osztályozásával foglalkozik az algoritmus, nem kezel vegyes rekordokat (pl. olyan rekordokat, ahol van egy idősor és több nem idősor jellegű adat). Bár ez az idősorok között általános, csak intervallum vagy arányskála típusú változóval rendelkező idősorokat kezel a módszer.

2.1.2. Szemtologató (EyeShifter)

Minden idősorhoz egy pointert rendelünk, ami az idősor egy elemére mutat. Ezt a pointert *szem*nek nevezzük. A *szemtologató* (*EyeShifter – ES*) feladata, hogy ezt a szemet mozgassa. Az egész algoritmus erről a pointermozgatásról kapta a nevét. A későbbi bővíthetőség érdekében a modul úgy került megalkotásra, hogy a szemtologató több *szemtologató-operátor*t (*EyeShifter Operator – ESO*) tartalmazzon. A modellépítés és az osztályozás során ez az egység eltérően működik. Az előbbinél az összes operátor lefut, az utóbbinál csak az, amit a csomópontban felépített modell tartalmaz. Erről bővebben a 2.2. és a 2.3. pontban írok.

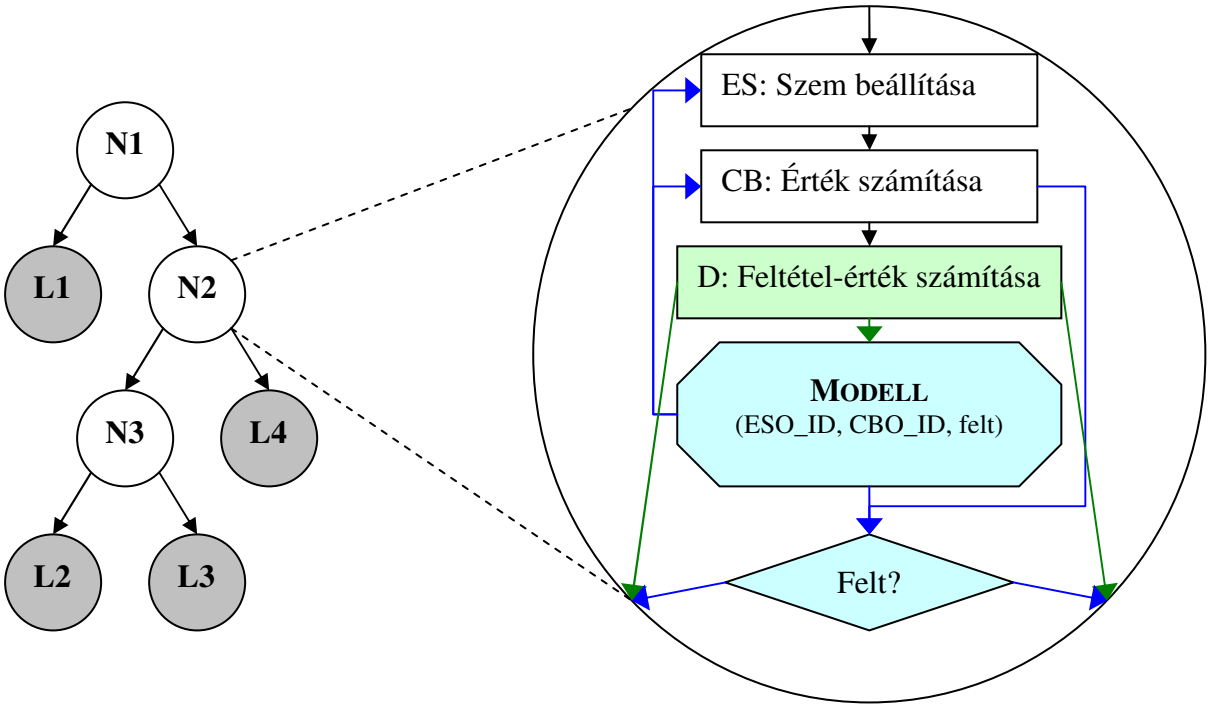
2.1.3. Feltételállító (ConditionBuilder)

A *feltételállító* (*ConditionBuilder – CB*) feladata, hogy a szem által mutatott pontbeli érték alapján egy származtatott értéket képezzen. Ezekre a számított értékekre fogalmazunk meg feltételeket a modellépítés során, és ezeket vesszük figyelembe az osztályozáskor. Úgy is fogalmazhatok, hogy a feltételállító minden idősorhoz a szem helyzete (és esetleges egyéb paraméterek) alapján egy értéket rendel, ami az adott idősor vágáskor figyelembe vett attribútumaként szerepel az adott csomópontban. Ez a számított érték helyettesíti azt az attribútumot, amit a klasszikus osztályozó egy klasszikus osztályozási feladatnál kiválasztana. A szemtologatóhoz hasonlóan ez a modul is több, úgynevezett *feltételállító-operátor*t (*ConditionBuilder Operator – CBO*) tartalmaz, és itt is az összes, illetve a kiválasztott fut le a modellezés és az osztályozás során. A modellezés során ez a modul egy extra feladatot is végrehajt: specifikálja az összes vágási helyet. Erről bővebben a 2.2. pontban írok.

2.1.4. Döntő (Decider)

A *döntő* (*decider*) feladata, hogy kiválassza, hogy a csomópontba kerülő tanítópontokat hogyan osszuk ketté, azaz válasszon a lehetséges vágások közül. Ez, ellentétben az előző kettő részegységgel, egy olyan modul, aminek csak a modellezés során van szerepe. A változtathatóság miatt itt is operátorokról van szó, de mivel a döntő mindig pontosan egy operátort tartalmaz, ezért ezeknek nincs külön neve. Az egyes operátorok abban térnek el, hogy milyen jóságértékkel operálnak, azaz milyen függvényt használnak a vágások rangsorának megállapítására. A modul ezek közül az értékek közül választja ki a legnagyobbat vagy legkisebbet (operátorfüggő), és az ahhoz tartozó vágást (feltétel-értéket).

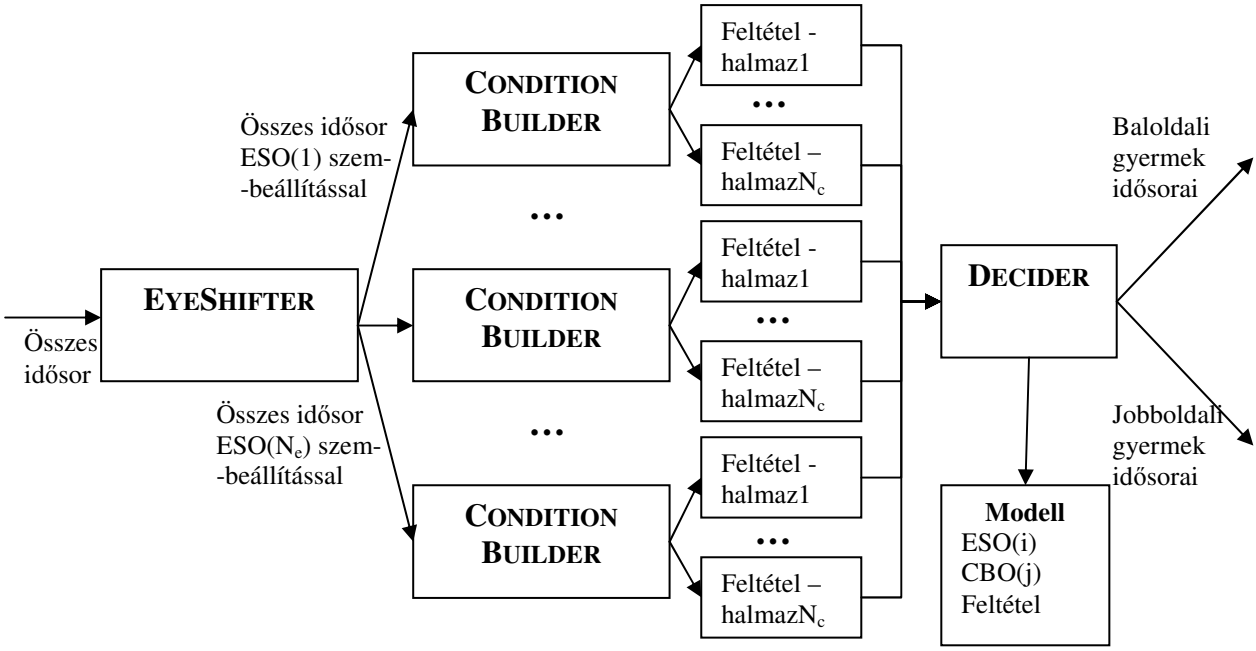
2.1.5. Egy csomópont kialakítása



2.1. ábra: A ShiftTree egy csomópontjának felépítése, a részegységek kapcsolata

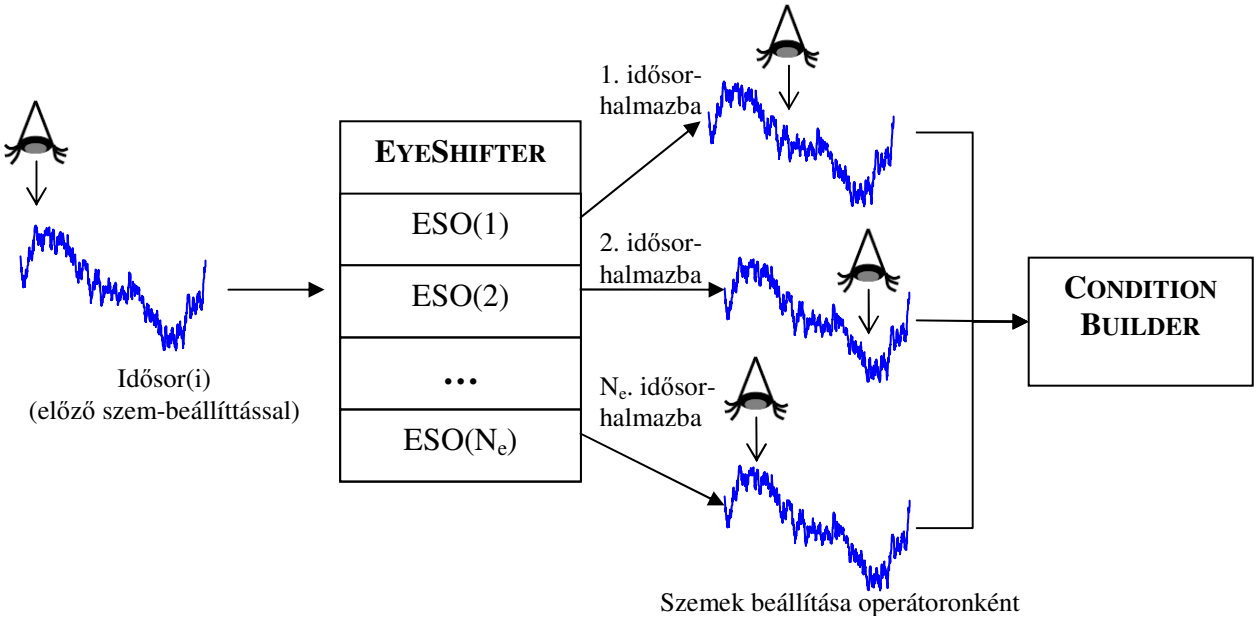
A 2.1. ábra mutatja, egy nem-levél csomópont felépítését. A zöld háttérrel rajzolt modulok és nyilak csak a modellezés, a kékkel színezettek csak az osztályozás során érvényesek. Fentebb nem szerepelt a modell, mivel nem részegység, de egyértelmű, hogy ezt tárolnunk kell a csomópontban. A „Felt?” jelölésű rombusz nem részegységet jelöl, hanem arra utal, hogy az osztályozásnál a modellben tárolt feltétellel egy összehasonlítást végzünk.

2.2. Az algoritmus tanítása (modellépítés)



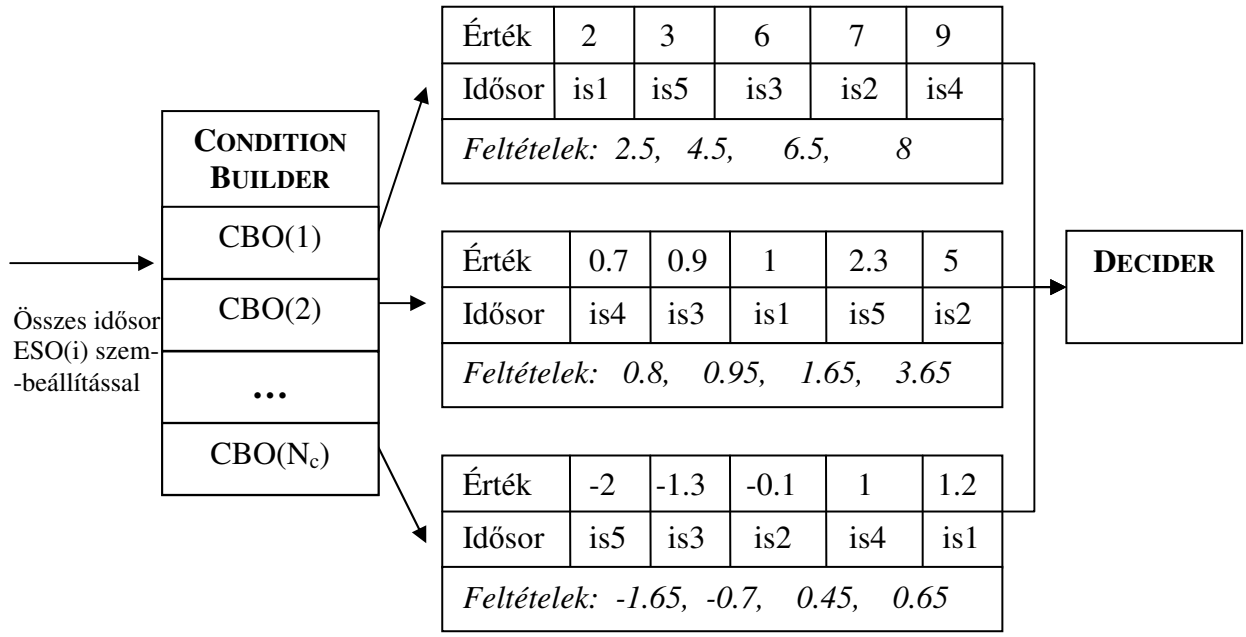
2.2. ábra: A ShiftTree egy csomópontjának működése modellezés közben

A 2.2. ábra mutatja egy csomópont működését a tanítás során. Minden csomópontba adott számú tanítópont (idősor) kerül. A csomópont összes idősorán az EyeShifter beállítja az összes operátora szerint a szemeket, ami N_e darab idősor-halmazt eredményez, ha N_e az ESO-k száma. Ezek a beállított idősor-halmazok sorban továbbmennek a ConditionBuilder-be, ami az összes operátora alapján előállít idősor-halmazonként N_c feltételhalmazt, ha N_c a CBO-k száma, és egyúttal definiál feltételhalmazonként *N–1* darab lehetséges vágási helyet, ha N a csomópont tanítópontjainak a száma. Az összes lehetséges, azaz $N_e * N_c * (N-1)$ darab lehetséges vágási hely továbbmegy a decider-be, ami kiválaszt közülük egyet. Azt választja, ami a maximális jóságértéket szolgáltatja. A hozzá tartozó ESO-t, CBO-t, és a feltétel értékét feljegyzi a csomópont modelljébe. A két részre vágott idősor halmazt a gyermek csomópontoknak adja tovább. Az idősoron a szem az eredeti helyhez képest a kiválasztott ESO hatásának megfelelően elmozdul.



2.3. ábra: A szemtologató működése a modellezés során

A 2.3. ábra mutatja a szemtologató működését a modellezés során. A bemenetén kapott idősor-halmaz minden idősorán a szemet minden operátora szerint beállítja, majd ezeket a különböző módon beállított halmazokat továbbítja a feltételállítónak. Ez természetesen az implementált verzióban nem azt jelenti, hogy fizikailag megsokszorozza az idősorokat, hanem először az első beállítással továbbküldi őket, majd ha a többi részegység végzett, akkor a 2. beállítással, és így tovább.



2.4. ábra: A feltételállító működése a modellezés során

A 2.4. ábra mutatja a feltételállító működését a modellezés során. Az egyes idősorok közti feltétel-értékek tájékoztató jelleggel szerepelnek az ábrán, mivel azokat nem a feltételállító, hanem a döntő számolja ki.

Minden CBO egy sorrendezést hajt végre az egyes idősor-halmazokon. A sorrendezés alapját az egyes idősorokra számított sorrendezési értékek képzik, amiket a CBO számol a szemek által mutatott értékekből. Ezzel a sorrendezéssel *N–1* feltételt állít, ha *N* az idősorok száma, mégpedig úgy, hogy a majdani vágási feltételek a szomszédos idősorok sorrendezési értékeinek átlagai. Azért csak a szomszédos idősorok között értelmezek vágási helyet, mert a döntési fák olyan feltételeket állítanak intervallum típusú attribútumok esetén, hogy az adott attribútum kisebb-e, mint egy érték. Ha van egy sorrendezés, akkor ha két szomszédos elem értéke között vág a módszer, akkor a baloldali elemet megelőző összes elemre a feltétel igaz, a jobboldalit követőekre hamis lesz. Egy CBO tehát, minden lehetséges vágási helyet definiál.

A feltételállító is hasonlóan működik, mint a szemtologató: az összes operátorát lefuttatja az idősorokon, ezáltal $N_c*(N–1)$ lehetséges vágási feltételt állít (szem-beállításonként), ha *N* a csomópontba került idősorok száma és N_c a feltételállító operátorok száma.

A decider tehát az összes lehetséges vágási helyet látja, és meg tudja határozni, hogy egy vágás hogyan bontaná két részre a tanítópontokat. Ezek alapján a benne definiált jóságértéket már ki tudja számolni. Ezek közül megkeresi a maximálisat, kiszámolja a vágás két szélén lévő származtatott érték átlagát, ami a vágási feltétel értéke, utasítja a csomópontot a modell aktualizálására, és továbbadja a két tanítóhalmazt a gyermek csomópontoknak. Az egyes csomópontokban felépített modellnek tartalmaznia kell, hogy milyen szem-beállítást kell végezni (ESO_ID), milyen CBO-t kell használni (CBO_ID) és, hogy milyen feltétel-értéket választott ki a decider (feltétel-érték). Ezen kívül szükség van a csomópontba a modellépítés során került idősorok osztályainak eloszlására, vagy a többségi osztály azonosítójára.

2.3. Az osztályozás menete a felépített modell alapján

Az osztályozásnál a feljegyzett modell alapján jár el az algoritmus: szem beállítása a modellben feljegyzett ESO alapján, sorrendezési érték származtatása a modellben szereplő CBO alapján. Ha a származtatott érték kisebb, mint a feltétel érték, akkor a bal gyermek felé, ha nagyobb vagy egyenlő, akkor a jobb gyermek felé továbbítja az idősort. Ha az idősor egy levél csomópontba ért, akkor visszaadja az adott levélben legnagyobb számmal képviselt osztály azonosítóját és/vagy annak arányát a levélbe került tanítópontokhoz képest.

2.4. Egy- és többváltozós ShiftTree-k összehasonlítása

A fent leírt módszer általános, és elvben nem szorítkozik csak az egyváltozós idősorok osztályozására. A pontosság kedvéért viszont szükség van annak leírására, hogy a többváltozós idősorok osztályozása milyen részletekben tér el az egyváltozós esettől.

Kiemelném, hogy a többváltozós osztályozásra több eltérő lehetőség lenne a módszer keretein belül. A legalapvetőbb kérdés, hogy mennyi legyen a szemek száma. Az első variáció szerint a szemek száma legyen továbbra is egy, azaz egy pointerünk van, amit tologatunk az idősor időtengelye mentén. A továbbiakban ezzel foglalkozok, de megemlítek egy másik lehetőséget is. Ez a lehetőség sokszemű ShiftTree, amiben annyi szemünk van, ahány változóval az idősor rendelkezik. Ezeket a szemeket egymástól függetlenül állíthatjuk. Fontos látni, hogy az egyes szemek ilyenkor is – az adott időpillanatban – az összes változó értékét látják. A sokszemű megoldás egyik jelentős hátránya – emiatt nem került megvalósításra – hogy jelentősen megnövelné a futási időt, hiszen minden lehetséges szembeállítás-kombinációt el kell végeznünk, ami $(N_e)^{N_v}$ a feltételállítóba továbbmenő idősor-halmazt jelent (N_e az ESO-k, N_v a változók száma). Ezen felül nem látszik az sem, jobb lenne az egyszemű megoldásnál, viszont könnyen lehet, hogy érzékeny lenne a túltanulásra (lásd: 5.2.2.3. pont).

2.4.1. Változtatások az EyeShifter szintjén

A szemtologató szintjén nincsen különösebb változás. Annyi változik, hogy az eredetileg definiált ESO-k közül a legtöbbel az összes változón be kell állítani a szemet, ami külön idősor-halmazt jelent a feltételállító szempontjából. Ennek az az oka, hogy pl. egy „ugorjunk a következő lokális maximumra” típusú ESO máshová állítja a szemet a különböző változók függvényein (és így az időtengelyen), mert azoknak – feltehetőleg – máshol vannak a lokális maximumaik. Mivel most már számít az is, hogy melyik változóról van szó, fel kell jegyezni a modellbe is, hogy melyik változón állítottuk be az ESO-t. Természetesen vannak olyan ESO-k is, amelyek minden változó esetében ugyanoda tolják a szemet. Ilyen pl. az „ugorjunk előre 50-et” operátor. Az egységesség kedvéért ezekhez is feljegyezzük, hogy melyik változón állítottuk be őket. Mivel itt mindegy, egységesen az 1. változót jegyezzük fel.

Ezen felül definiálhatunk többváltozós ESO-kat, amelyek 2 vagy több változó eltérése, hasonlósága alapján tolják el a szemet egy adott helyre.

2.4.2. Változtatások a ConditionBuilder szintjén

A feltételállító szintjén két változás történik. Egyrészt az egyváltozós CBO-knak az ESO-khoz hasonlóan meg kell adni, hogy melyik változón végezzék el a számításokat, azaz minden CBO-t minden változón külön lefuttatunk, ami külön feltételhalmazokat generál a decidernek. Nyilvánvaló, hogy erre szükség van, mert más-más változókon egy adott CBO más-más eredményeket ad. Itt nincs olyan operátor, aminél ez megspórolható lenne. Fontos megjegyezni, hogy ez a változó független attól, amelyiken a szemet állítottuk be. Ezért ezt külön fel kell jegyezni a modellbe.

2.4.2.1. Új entitás: Feltételállító-segéd (ConditionBuilderExtension)

A másik változtatás a *feltételállító-segéd* (*ConditionBuilderExtension – CBE*) entitás bevezetése. Ez az entitás CBO-ként működik olyan szempontból, hogy idősoronként egy származtatott értéket ad vissza, és feltételhalmazokat generál. Az eltérés abban keresendő, hogy létrehozásakor meg kell adni neki egy egyváltozós CBO-t. Ő ezzel a CBO-val kiszámolja a származtatott értéket az idősor minden változójára, majd a típusának megfelelően ezekből a származtatott értékekből számítja ki a visszaadott származtatott értéket. Egy példa: a CBO-nk legyen olyan, hogy az adott változón a szem helyének megfelelő értéket adja vissza. A CBE legyen átlag típusú, ami azt jelenti, hogy a CBO-tól kapott származtatott értékek átlagát veszi. Ezzel az operátor összeállítással a szem által mutatott értékek átlagát kapja meg az idősor, mint származtatott értéket.

2.5. Az implementációban definiált operátorok

A továbbiakban bizonyos részek megértéséhez fontos lehet ismerni azokat az operátorokat, amiket az implementált változat tartalmaz. Ezek természetesen csak egy kis részét teszik ki a lehetséges operátoroknak, de mint látható lesz, már ezekkel is jó hatékonyság érhető el.

2.5.1. Szemtologató-operátorok

- ESONext(X) - ugrás előre a szem helyéhez képest X hellyel, vagy az idősor végére, ha az korábban véget ért
- ESOPrev(X) - ugrás vissza a szem helyéhez képest X hellyel, vagy az idősor elejére, ha túllépne az idősor elején
- ESONextMax - ugrás a szem helyétől számított következő lokális maximumhoz, ha nincs ilyen, akkor egyhelyben marad
- ESOPrevMax - ugrás a szem helyétől számított előző lokális maximumhoz, ha nincs ilyen, akkor egyhelyben marad
- ESONextMin - ugrás a szem helyétől számított következő lokális minimumhoz, ha nincs ilyen, akkor egyhelyben marad
- ESOPrevMin - ugrás a szem helyétől számított előző lokális minimumhoz, ha nincs ilyen, akkor egyhelyben marad
- ESOMax - ugrás a globális maximumhoz
- ESOMin - ugrás a globális minimumhoz

2.5.2. Feltételállító-operátorok

- CBOSimple - a szem által mutatott pontbeli érték visszaadása
- CBONormal(M,V,K) - a szem környezetének (M,V) paraméterű normális eloszlás szerinti súlyozott átlaga (mindkét irányban), ahol a minimális súly K:

  $$\frac{\sum_{t=-T}^{T} \frac{1}{V\sqrt{2\pi}} * e^{\frac{-(|t|-M)^2}{2V^2}} * x_t}{\sum_{t=-T}^{T} \frac{1}{V\sqrt{2\pi}} * e^{\frac{-(|t|-M)^2}{2V^2}}}$$

  , ahol M az eloszlás várható értéke, V az eloszlás szórása, t a szem helyétől való eltérés az időtengelyen, x_t a változó értéke a t helyen és teljesül, hogy $\frac{1}{V\sqrt{2\pi}} * e^{\frac{-(|t|-M)^2}{2V^2}} \geq K$.

- CBOExp(L,K) - a szem környezetének L paraméterű exponenciális eloszlás szerinti súlyozott átlaga (mindkét irányban), ahol a minimális súly K:
$\frac{\sum_{t=-T}^{T} L * e^{-L*|t|} * x_t}{\sum_{t=-T}^{T} L * e^{-L*|t|}}$, ahol L az eloszlás várható paramétere, t a szem helyétől való eltérés az időtengelyen, x_t a változó értéke a t helyen és teljesül, hogy és $L * e^{-L*|t|} \geq K$.
- CBOAvg(L) - a szem L sugarú környezetének átlaga:
$\frac{\sum_{t=-L}^{L} x_t}{2L+1}$, ahol L a megadott sugár és x_t a változó értéke a t helyen.
- CBEAverage(CBO) - a megadott CBO változónkénti származtatott értékeiből értékekből átlagot számít
- CBEVariance(CBO) - a megadott CBO változónkénti származtatott értékeiből értékekből szórást számít

2.5.3. Decider típusok

- DEntropy - entrópia alapú jóságérték számítás [9]:
$\frac{N_B}{N} * \sum_{i=1}^{N_C} - P_{Bi} * \log_2 P_{Bi} + \frac{N_J}{N} * \sum_{i=1}^{N_C} - P_{Ji} * \log_2 P_{Ji}$,
ahol P_{Bi}, P_{Ji}, P_i a bal-, a jobb gyermekben, és a szülőben az i. osztály relatív gyakorisága, N_B, N_J, N a tanítópontok száma, N_C az osztályok száma
- DGINI - GINI index alapú jóságérték számítás [10]:
$\frac{N_B}{N} * \left(1 - \sum_{i=1}^{N_C} {P_{Bi}}^2\right) + \frac{N_J}{N} * \left(1 - \sum_{i=1}^{N_C} {P_{Ji}}^2\right)$,
ahol P_{Bi}, P_{Ji}, P_i a bal-, a jobb gyermekben, és a szülőben az i. osztály relatív gyakorisága, N_B, N_J, N a tanítópontok száma, N_C az osztályok száma
- DDivFunc(L) - súlyozott divergenciafüggvény [11] alapú jóságérték számítás:
$\frac{1}{L*(L-1)} * \left(\frac{N_B}{N} * \sum_{i=1}^{N_C} P_{Bi} * \left(\left(\frac{P_{Bi}}{P_i}\right)^L - 1\right) + \frac{N_J}{N} * \sum_{i=1}^{N_C} P_{Ji} * \left(\left(\frac{P_{Ji}}{P_i}\right)^L - 1\right)\right)$,
ahol P_{Bi}, P_{Ji}, P_i a bal-, a jobb gyermekben, és a szülőben az i. osztály relatív gyakorisága, N_B, N_J, N a tanítópontok száma, N_C az osztályok száma

3. A futási idő kérdése

Az algoritmus elméleti alapjait az előző fejezetben láthattuk. Mielőtt rátérnék a módszer tesztelésére, egy nagyon fontos kérdést, a futási időt veszem górcső alá. Ebben a fejezetben kiszámolom az egyattribútumos ShiftTree futási idejét, majd megmutatom, hogy a többváltozós változatra áttérés mennyivel növeli meg azt.

3.1. Egyattribútumos idősorokat osztályozó ShiftTree futási ideje

A teljes futási időt úgy kaphatjuk meg, hogy kiszámoljuk csomópontonként, és utána ezeket összegezzük az egész fára. Ebben a pontban először egy adott mennyiségű tanítóponttal rendelkező csomópont futási idejét számolom ki, majd ennek alapján megadom a teljes fa futási idejét. A pont végén példákon keresztül megvizsgálom, hogy milyen hatással van a fa struktúrájának változtatása a futási időre.

3.1.1. Csomópontonkénti futási idő

$T_{futás} = \sum_{i=1}^{N_{ESO}} T_{ESOi} * N * \left(\sum_{j=1}^{N_{CBO}} T_{CBOj} \right) * T_D$, ahol N_{ESO} az ESO-k, N_{CBO} a CBO-k, N a csomópont idősorainak száma, T_{ESOi} az i. ESO, T_{CBOj} pedig a j. CBO átlagos futási ideje egy idősoron és T_D a decider futási ideje.

A futási idő képlete úgy jön ki, hogy minden csomópontban az algoritmus az összes idősort az összes szem szerint beállítja, majd ezeket az összes CBO szerint megvizsgálja, és az egészen végigfuttatja a decidert. Ezen műveletek idejének szorzataként jön ki a teljes futási. Az ESO-k és a CBO-k futási időit összegezni kell, mielőtt összeszoroznánk őket a többi taggal.

A képlet egy kicsit átalakítva, kiemelve az egyszerű szorzó tényezőket a szummák elé így néz ki: $T_{futás} = N * T_D * \sum_{i=1}^{N_{ESO}} T_{ESOi} * \left(\sum_{j=1}^{N_{CBO}} T_{CBOj} \right)$

Az egyes operátorok futási idői az adatok több tulajdonságától függhetnek. Megvizsgáltam, hogy mitől függenek az egyes operátortípusok futási idői.

- Az ESO-k futási ideje olyan módon függ az idősoroktól, ami arányossággal nem kifejezhető. Egy (lokális) minimum vagy maximumkeresés nem csak adatsoronként, de az egyes idősorok esetében is eltérő ideig tarthat. Átlagosan viszont igaz, hogy a hosszabb idősorokon a folyamat hosszabb lesz. Az X hellyel arrébb ugrás minden idősoron ugyanannyi ideig tart egészen addig, amíg az ugrás túl nem ér valamely idősoron (mert akkor rövidebb az ugrás). Emiatt itt is átlagosan a hosszabb idősorokon tart tovább az ugrás, de kisebb X számoknál egyenlő ideig tart. A fentiek miatt mondhatom azt, hogy $T_{ESOi} = O(L_{TS})$, ahol L_{TS} a csomópont idősorainak hossza. De ennek ellenére, a pontatlanságok (nem minden esetben van meg egyértelműen ez a függés) és jelentéktelensége miatt ezt a tagot egy adott ShiftTree-re jellemző specifikus szorzónak tekintem a továbbiakban.
- A CBO-k futási ideje az idősorok számától függ, mivel minden CBO minden idősorhoz a szem-beállítás alapján egy értéket számol. Azaz $T_{CBOi} = N * t_{CBOi}$, ahol t_{CBOi} az i. típusú CBO által egy idősoron a származtatott érték kiszámításához szükséges idő. Ez utóbbi lényegében idősor független (bizonyos esetekben egyes idősorokon túllóghatunk a súlyozás során, amit ekkor abbahagyunk, de ez ritka, és nem jelentős időmegtakarítással járó esemény).

- A decider időigénye szintén függ az idősorok számától, mert minden lehetséges vágási helyhez egy jóságértéket számol. A lehetséges vágási helyek a sorba rendezett idősorok közti helyek, tehát *N–1* hely. Azaz $T_D = (N-1)*t_D$, ahol t_D egy lehetséges vágási helyen a jóságérték kiszámolásának ideje. Ez utóbbi nem függ az adatsor tulajdonságaitól, teljesen decider specifikus (pl.: az entrópia alapú decider a logaritmus miatt tovább számol, mint a GINI index alapú).

A fenti képlet ezek alapján: $T_{futás} = N^2 * (N-1) * t_D * \sum_{i=1}^{N_{ESO}} T_{ESOi} * \left(\sum_{j=1}^{N_{CBO}} t_{CBOj} \right)$

Ahonnan a két szummát nyugodtan helyettesíthetjük egy, a fát jellemző szorzóval T_{ST}-vel: $T_{futás} = N^2 * (N-1) * t_D * T_{ST}$

3.1.2. A teljes ShiftTree futási ideje:

A csomópontokra összegezzük a futási időket: $T_{futás} = \sum_{i=1}^{N_{node}} N_i^2 * (N_i - 1) * t_D * T_{ST}$, ahol N_{node} a csomópontok száma, T_{ST} a fára jellemző szorzó tényező, N_i pedig a tanítópontok száma az i. csomópontban.

Kiemelve a szorzó tagokat: $T_{futás} = t_D * T_{ST} * \sum_{i=1}^{N_{node}} N_i^2 * (N_i - 1)$

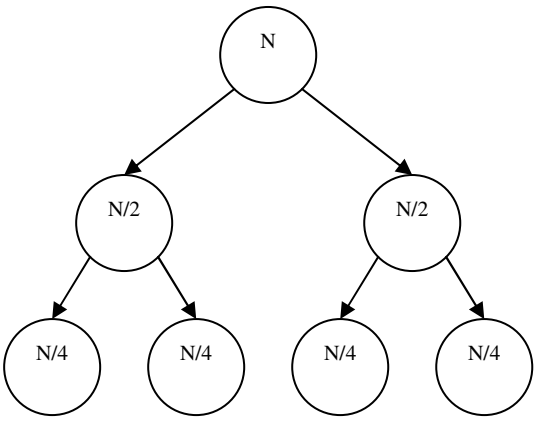
Innentől kezdve csak a szumma utáni résszel foglalkozok. Ez a rész eléggé általános, de szerencsére van pár megkötésünk, amik a bináris döntési fa jellegből adódnak:

- Egy szülő csomópontnak vagy 0 vagy 2 gyermeke van, más eset nem fordulhat elő.
- Minden csomópontban pozitív egész számú tanítópont van.
- Egy szülő csomópont gyermekeiben a tanítópontok összessége megegyezik a szülő tanítópontjainak számával.
- Minden olyan csomópont, amelyben csak egy elem van, biztos, hogy levél.

A megkötések ellenére még mindig túl összetett a probléma, ami miatt általános esetre nem lehet zárt alakban képletet felírni. Ezért inkább néhány speciális helyzetre számoltam ki az összeget, és ebből vontam le következtetéseket a futási időre vonatkozón.

3.1.3. Példák eltérő szerkezetű fák építési idejére

3.1.3.1. Teljes fa, felezés csomópontonként (1. példa)



3.1. ábra: Teljes fa tanítópont felezéssel

A 3.1. ábrán látható módon elrendezett fára összegezem a csomópontok futási idejét.

A fában minden csomópontnak pontosan 2 gyermeke van.

A tanítópontok száma feleződik a szülő gyermekei között.

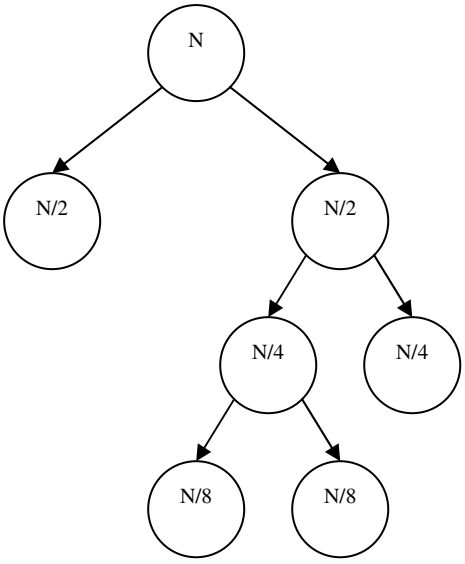
Az alábbi módon számolható a szumma értéke:

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^2(N-1) + 2\left(\frac{N}{2}\right)^2\left(\frac{N}{2} - 1\right) + 4\left(\frac{N}{4}\right)^2\left(\frac{N}{4} - 1\right) + \ldots = N^3 - N^2 + \frac{N^3}{4} - \frac{N^2}{2} + \frac{N^3}{16} - \frac{N^2}{4} + \ldots$$

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^3\left(1 + \frac{1}{4} + \frac{1}{16} + \ldots\right) - N^2\left(1 + \frac{1}{2} + \frac{1}{4} + \ldots\right) = N^3 \frac{\left(\frac{1}{4}\right)^d - 1}{\frac{1}{4} - 1} - N^2 \frac{\left(\frac{1}{2}\right)^d - 1}{\frac{1}{2} - 1}$$

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = 4N^3 \frac{1 - \left(\frac{1}{4}\right)^d}{3} - 2N^2\left(1 - \left(\frac{1}{2}\right)^d\right)$$, ahol d a szintek száma (levelek szintjét nem számolva); d maximális értéke $d_{\max} = \lfloor \log_2 N \rfloor - 1$

3.1.3.2. Szintenkénti egy vágás, felezés csomópontonként (2. példa)



A 3.2. ábrán látható módon elrendezett fára összegezem a csomópontok futási idejét.

A fában minden szinten pontosan egy olyan csomópont van, aminek van 2 gyermeke. Az összes többi csomópont levél.

A tanítópontok száma feleződik a szülő gyermekei között.

Ez a szerkezet a legkisebb futási idejű eset, mivel ilyen struktúrában van a legkevesebb csomópont (szintenként 2, kivéve a gyökérnél), valamint a tanítópontok feleződésével minimalizáljuk az $N^2 * (N-1)$ kifejezések összegét szintenként.

3.2. ábra: Minimális fa tanítópont felezéssel

A fenti állítás bizonyítása:

$x = a + (x - a)$ → vágás x pontot tartalmazó csomópontnál a és x-a méretű részekre

$f(a) = a^2(a-1) + (x-a)^2(x-a-1)$ → a két gyermek csomópont szummájának összege

$f(a) = -2a^2 + x^3 - 3ax^2 + 3a^2x - x^2 + 2ax$

$f'(a) = -4a - 3x^2 + 6ax + 2x = 0$ → szélsőértékek keresése

$4a - 6ax = 2a(2 - 3x) = x(2 - 3x) = 2x - 3x^2$

Mivel x egész, $(2-3x)$ biztos, hogy nem 0, tehát: $a = \frac{x}{2}$, a felezés adja a szélsőértéket

$f''(a) = -4 + 6x > 0$,mivel $x \geq 1$ → a kiszámított érték a kifejezés minimumát adja.

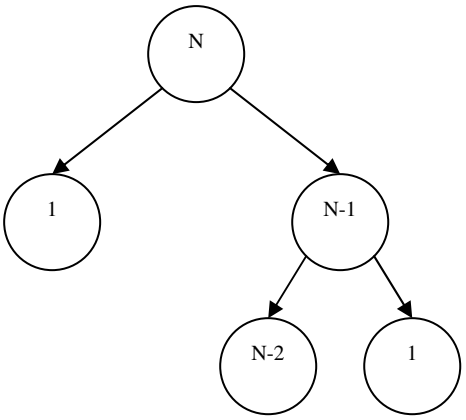
Az alábbi módon számolhatom ki az összeg értékét:

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^2 (N-1) + 2\left(\frac{N}{2}\right)^2 \left(\frac{N}{2} - 1\right) + 2\left(\frac{N}{4}\right)^2 \left(\frac{N}{4} - 1\right) + \ldots = N^3 - N^2 + \frac{N^3}{4} - \frac{N^2}{2} + \frac{N^3}{32} - \frac{N^2}{8} + \ldots$$

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^3 \left(1 + \frac{1}{4} + \frac{1}{32} + \ldots\right) - N^2 \left(1 + \frac{1}{2} + \frac{1}{8} + \ldots\right) = N^3 \left(1 + \frac{1}{4} \frac{\left(\frac{1}{8}\right)^{d-1} - 1}{\frac{1}{8} - 1}\right) - N^2 \left(1 + \frac{1}{2} \frac{\left(\frac{1}{4}\right)^{d-1} - 1}{\frac{1}{4} - 1}\right)$$

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^3 \frac{9 - 2 * \left(\frac{1}{8}\right)^{d-1}}{7} - N^2 \frac{5 - 2 * \left(\frac{1}{4}\right)^{d-1}}{3}$$, ahol d a szintek száma (levelek szintjét nem számolva); d maximális értéke $d_{\max} = \lfloor \log_2 N \rfloor - 1$

3.1.3.3. Szintenkénti egy vágás, egy pont kivétele csomópontonként (3. példa)



A 3.3. ábrán látható fában minden szinten pontosan egy olyan csomópont van, aminek van 2 gyermeke. Az összes többi csomópont levél.

Minden nem-levél csomópont úgy osztja ketté a tanítópontokat, hogy az egyik gyermekébe 1, a másikba n-1 pont kerül.

Ez az ilyen szerkezetű fáknál a legnagyobb számításigényű szétosztás, mivel az előző számításnál bemutatott f(a)-nak nincs lokális maximuma és egy lokális minimuma van az intervallum felénél, tehát a maximumát az intervallum szélein veszi fel.

3.3. ábra: Minimális fa tanítópontok csökkentésével

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^2 (N-1) + (N-1)^2 (N-2) + (N-2)^2 (N-3) + (N-3)^2 (N-4) + \ldots$$

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = \left(N^3 - N^2\right) + \left(N^3 - 4N^2 + 5N - 2\right) + \left(N^3 - 7N^2 + 16N - 12\right) + \left(N^3 - 10N^2 + 33N - 36\right) + \ldots$$

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^3 d - N^2 (1 + 4 + 7 + 10 + \ldots) + N(0 + 5 + 16 + 33 + \ldots) - (0 + 2 + 12 + 36 + \ldots)$$

$$\sum_{i=1}^{N_{node}} {N_{TSi}}^2 (N_{TSi} - 1) = N^3 d - N^2 \left(\frac{(3d-1)d}{2}\right) + N\left(\frac{(d-1)(2d+1)d}{2}\right) - \left(\frac{(d-1)d(3d-2)(d+1)}{12}\right)$$, ahol d a szintek száma (levelek szintjét nem számolva); d maximális értéke $d_{\max} = N - 1$

3.1.4. Következtetés a példák alapján

A példák áttekintése után látható, hogy a futási idő nagyban függ a fa struktúrájától, a tanítópontok eloszlásától és a fa mélységétől. Ez a három dolog legalább akkora mértékben határozza meg a futási időt, mint az idősorok száma. A tanítópontok eloszlása nagy hatással van arra, hogy a szintek száma milyen mértékben szól bele a futási időbe (2. és 3. példa). A hasonló ponteloszlás, eltérő struktúra mellet, az együtthatókat befolyásolja (1. és 2. példa). Az is megfigyelhető, hogy a futási idő egyáltalán nem függ se az osztályok számától és az idősorok hosszától se.

3.2. Többattribútumos idősorokat osztályozó ShiftTree futási ideje

A többattribútumos változatnál az okoz futási idő többletet, hogy mind az ESO-kat, mind a CBO-kat az összes változón be kell állítani. Tehát a futási időt a változók száma befolyásolja, ami egy szorzó tényezőben nyilvánul meg mind csomópont, mind fa szinten. A decider futási idejét a több változó nem befolyásolja, mivel ő csak a származtatott értékekkel foglalkozik, az eredeti változókkal nem. A származtatott értékekből viszont mindig csak egy van idősoronként egy adott ESO és CBO beállítás mellett.

Az ESO-k esetében csak a definiált operátorok ismeretében mondhatunk bármit arra vonatkozóan, hogy hányszorosára növekszik a futási idő, mivel az „ugorjunk előre/vissza X helyet" (ESONext/ESOPrev) típusú operátorokat elég egyetlen változón beállítani. Hiszen az ilyen típusú operátorok hatása ugyanaz, bármelyik változón is alkalmazzuk őket. Ezért van az, hogy a modellépítés idejének növekedése az egyváltozós esethez képest a definiált ESO-k típusától függ. Ha minden ESO ESONext/ESOPrev jellegű, akkor az operátorok beállítása az szemtologató szintjén nem növeli a futási időt. Ha egyik ESO sem ilyen, akkor a futási idő ezen a szinten változószám-szorosára nő.

A feltételállító szintjén az egyattribútumos CBO-kat minden változón le kell futtatni. A CBE-vel definiált operátorok igaz ugyanazt az egy értéket szolgáltatják az összes változó esetén, és ezért csak egyszer futnak le, de ehhez az értékhez a bennük definiált CBO-t minden változón le kell futtatni. Emiatt ezen a szinten a futási idő mindig pontosan az eredeti változószám-szorosára nő.

Összességében tehát az mondható, hogy $T_{többváltozós} = N_v{}^2 * \frac{N_{ESONoJump}}{N_{ESO}} * T_{egyváltozós}$, ahol N_V az idősor változóinak a száma, $N_{ESONoJump}$ a nem ESONext/ESOPrev jellegű, N_{ESO} pedig az összes ESO száma.

3.2.1. Sokszemű ShiftTree futási ideje

A 2.4. pontban említettem, hogy a sokszemű ShiftTree futási ideje jelentősen növekszik, és ennek okán lett elvetve a kidolgozása. Ebben a pontban ezt részletesebben is megindoklom.

Korábban látható volt, hogy a többszemű esetnél az ESO-k minden egyes kombinációját be kell állítani az egyes változókon. Tehát 2 változó és 2 ESO esetén le kell futtatni az 1-1, az 1-2, a 2-1 és 2-2 beállításokat is, és ezeket így továbbadni a feltételállítónak. Jól látható, hogy ez a feltételállító bemenetét és a szemtologató futási idejét a $(N_{ESO})^{N_v}$ -szorosára növeli.

A feltételállító szintjén nem történik változás az egyszemű esethez képest, tehát ott a futási idő az eredeti N_v-szeresére nő.

Összegezve a fentieket, a futási idő az alábbi módon kapható meg:

$$T_{sokszemű} = (N_{ESO})^{N_v} * N_v * T_{egyváltozós}$$

Mivel az ESO-k száma elég nagy lehet (a tesztelésnél ezzel a kevés definiált operátorral 22 ESO-t hoztam létre), és az attribútumok száma legalább 2, látható, hogy ez a megoldás túl sok ideig futna.

4. Tesztelés egyattribútumos idősorokkal

Az algoritmus C++ nyelvű implementációját részletes tesztnek vetettem alá, hogy kiderüljön, hogy hogyan teljesít a gyakorlatban. Külön foglalkoztam az egyváltozós és a többváltozós esettel. Ebben a fejezetben az előbbiről lesz szó.

A fejezet elején ismertetem azokat az adatokat, amiket a teszteléshez használtam. Utána rátérek arra, hogy más algoritmusokkal összehasonlítva hogyan teljesít a ShiftTree. Mivel fontos cél volt számomra a modellek értelmezhetősége, a következő alfejezet ezzel foglalkozik. Végül a módszer mélyebb jellemzőit kutatva egy alfejezet keretében keresztvalidációs méréseket végzek a futási időnek és pontosságnak a tanítópontok számától való függésének megállapítására.

4.1. A teszteléshez használt adatok

Ebben a pontban bemutatom azokat az adatsorokat, amiket a teszteléshez használtam. Ez tartalmaz egy nemzetközi benchmark adatbázist és egy osztályozási verseny adatsorát.

4.1.1. Nemzetközi benchmark adatsorok

Mivel fontosnak tartottam, hogy az eredmények összehasonlíthatóak legyenek más módszerek eredményeivel, a tesztelések egy részéhez nemzetközi benchmark adatsorokat [12] használtam. 20 különböző területről származó adathalmazzal dolgoztam. Az adatsorokhoz többféle elterjedt módszer pontossága elérhető. Az adatsorokban annyi a közös, hogy mindegyik olyan idősorokat tartalmaz, melyek egyváltozósak, az idősorok hossza adatsoronként megegyezik és az adatok z normalizáltak.

Az adatsorok tulajdonságait mutatja a 4.1. táblázat.

Adatsor	Osztályok száma	Tanítóhalmaz mérete	Teszthalmaz mérete	Idősor hossza
50Words	50	450	455	270
Adiac	37	390	391	176
Beef	5	30	30	470
CBF	3	30	900	128
Coffee	2	28	28	286
ECG	2	100	100	96
FaceAll	14	560	1690	131
FaceFour	4	24	88	350
FISH	7	175	175	463
Gun_Point	2	50	150	150
Lighting2	2	60	61	637
Lighting7	7	70	73	319
OliveOil	4	30	30	570
OSULeaf	6	200	242	427
SwedishLeaf	15	500	625	128
synthetic_control	6	300	300	60
Trace	4	100	100	275
Two_Patterns	4	1000	4000	128
Wafer	2	1000	6164	152
Yoga	2	300	3000	426

4.1. táblázat: A benchmark adatsorok tulajdonságai

4.1.2. Egyéb egyváltozós adatok

Az előző pontban említett adatsorok nagy részének az egyik hátránya, hogy kevés tanítópontot tartalmaznak. A ShiftTree, mint általában a döntési fák, tanulás-intenzív módszer, azaz sok tanítómintára van szüksége ahhoz, hogy pontos legyen. A fenti adatsorok másik hátránya, hogy z normalizáltak, ami által sok esetben információ veszhet el, ráadásul az adatsorok már nem tekinthetőek a valós életből vett idősoroknak.

Emiatt két másik egyváltozós adatsort is használtam a tesztekhez, ezek egy osztályozási verseny, a Ford Classification Challenge [13] adatsorai: Ford_A és a Ford_B. Mindkét adatsorban egy adott alkatrészről gyűjtött mérési eredmények alkotnak egy idősort, és azt kell eldöntenünk, hogy az alkatrész hibás-e vagy sem. Mindkét adatsor 500 hosszú idősorokat tartalmaz. A különbség a két adatsor között, hogy a Ford_A adatsor adatain minimális a zaj, míg a Ford_B valós mérések eredményeit tartalmazza.

4.2. A módszer összehasonlítása más algoritmusok eredményeivel

Ez az alfejezet arról szól, hogy a ShiftTree más algoritmusokkal összehasonlítva mennyire tekinthető pontos idősor-osztályozónak.

4.2.1. A program konfigurációja

A futtatások során ugyanazt a szemtologató és feltételállító konfigurációt alkalmaztam, viszont minden mérést 11 különböző decider-rel végeztem el. A programban definiált operátorok leírása a 2.5. pontban található.

Decider-ek:	Entropy, GINI, DivFunc(-2/3), DivFunc(-1/2), DivFunc(-1/3), DivFunc(0), DivFunc(1/3), DivFunc(1/2), DivFunc(2/3), DivFunc(1), DivFunc(2)
ESO-k (sorrendben):	NextMax, NextMin, Next(1), Next(5), Next(10), Next(25), Next(50), Next(100), Next(200), Next(400), Max, Min, Prev(1), Prev(5), Prev(10), Prev(25), Prev(50), Prev(100), Prev(200), Prev(400), PrevMax, PrevMin
CBO-k (sorrendben):	Simple, Normal(0, 1, 0.05), Exp(1, 0.05), Normal(0, 0.5, 0.01), Exp(0.5, 0.01), Normal(0.5, 4, 0.01), Exp(2, 0.01)

4.2.2. A mérés menete

A mérés során minden decider-rel minden adatsoron 40 mérést végeztem. Ez úgy jön ki, hogy 8 különböző szintkorlát értéket használtam, ezek: 3, 5, 7, 9, 11, 13, 15, NINCS. Mindegyik szintkorlátos beállítást lefuttattam egy adott információnyereségi korlát mellett. Ez azt jelenti, hogy egy csomópont csak akkor végez el egy vágást, ha az egy adott információnyereségnél nagyobb értéket ad. Ez azt jelenti, hogy a két gyermek csomópont entrópiájának átlagából levonom a szülő csomópont entrópiáját. Ezt képlet szerint így számolom:

$$\frac{\sum_{i=1}^{N_C} p_{Bi} \log_2 p_{Bi} + \sum_{i=1}^{N_C} p_{Ji} \log_2 p_{Ji}}{2} - \sum_{i=1}^{N_C} p_i \log_2 p_i$$,ahol p_{Bi} a bal, p_{Ji} a jobb, p_i pedig a szülő csomópontban az i. osztály relatív gyakorisága, N_C pedig az osztályok száma.

A pontosságot a találati aránnyal számolom, azaz a helyesen osztályozott idősorok számát elosztom az összes osztályozandó idősorral. Az összes adat eredetileg fel volt osztva tanító és teszthalmazra (a Ford adatsoroknál volt egy validációs halmaz is), ezeket a felosztásokat használtam.

4.2.3. Eredmények és értékelésük

A különböző döntőknél kiválasztottam a legjobb eredményt minden egyes adatsorra, ezek láthatóak a 4.2. táblázatban.

Adatsor	GINI	Ent	Div 2	Div 1	Div 2/3	Div 1/2	Div 1/3	Div 0	Div-1/3	Div-1/2	Div-2/3
FordA	76,67%	79,39%	76,97%	76,67%	76,67%	77,27%	76,97%	79,39%	80%	82,73%	84,55%
FordB	70,91%	70,91%	72,12%	70,91%	69,39%	69,39%	73,33%	70,91%	71,82%	75,15%	74,55%
50Words	41,32%	35,82%	34,95%	41,32%	41,32%	39,78%	37,58%	35,82%	12,53%	12,53%	12,53%
Adiac	48,34%	48,85%	43,73%	46,04%	52,69%	51,41%	47,06%	48,34%	2,05%	2,05%	2,05%
Beef	33,33%	56,67%	46,67%	36,67%	53,33%	56,67%	56,67%	56,67%	33,33%	33,33%	33,33%
CBF	89,22%	89,22%	89,22%	89,22%	89,22%	89,22%	89,22%	89,22%	46,11%	46,11%	46,11%
Coffee	67,86%	67,86%	67,86%	67,86%	67,86%	67,86%	67,86%	67,86%	67,86%	67,86%	67,86%
ECG200	80%	82%	78%	80%	80%	79%	82%	82%	84%	84%	85%
FaceAll	62,07%	63,02%	56,04%	65,44%	57,81%	55,56%	58,99%	63,14%	7,75%	7,75%	7,75%
Face Four	47,73%	47,73%	71,59%	71,59%	47,73%	47,73%	47,73%	47,73%	21,59%	21,59%	21,59%
FISH	64%	66,86%	50,86%	55,43%	46,86%	48%	48%	66,86%	26,86%	26,86%	26,86%
Gun Point	79,33%	79,33%	75,33%	79,33%	79,33%	79,33%	79,33%	79,33%	79,33%	79,33%	79,33%
Lighting2	72,13%	77,05%	68,85%	72,13%	72,13%	72,13%	72,13%	77,05%	65,57%	65,57%	65,57%
Lighting7	54,79%	63,01%	61,64%	58,9%	54,79%	54,79%	56,16%	57,53%	30,14%	30,14%	30,14%
OliveOil	80%	80%	70%	70%	80%	80%	80%	80%	56,67%	56,67%	56,67%
OSULeaf	46,28%	50,83%	56,2%	45,04%	51,65%	52,89%	52,48%	50,83%	34,3%	34,3%	34,3%
Swedish Leaf	69,44%	65,92%	60,32%	64,96%	66,56%	70,4%	67,68%	66,88%	11,52%	11,52%	11,52%
synthetic control	92,33%	92,33%	86%	93,67%	92,33%	92,33%	92,33%	92,33%	47,67%	47,67%	47,67%
Trace	98%	100%	96%	98%	100%	100%	100%	100%	46%	46%	46%
Two Patterns	92,45%	94,18%	91,9%	92,45%	93,7%	93,55%	93,3%	94,18%	49,28%	49,28%	49,28%
wafer	97,58%	97,91%	95,64%	97,58%	98,25%	98,25%	98,02%	97,91%	97,99%	97,99%	97,99%
yoga	68,93%	67,7%	71,87%	69,77%	69,67%	70,33%	70,33%	67,7%	76,03%	75,93%	75,4%

4.2. táblázat: Különböző decider-ek legjobb pontosságai

Minden adatsornál zöld háttérrel emeltem ki a maximum pontosságot. Mint az várható volt, az egyes adatsorok nagyon eltérően viselkednek. Lényegében semmilyen szabály nem mondható arra, hogy abszolút értelemben melyik decider lenne a legjobb. Annyi látszik, hogy az entrópia sok esetben a legpontosabb, ezért érdemes lehet elsőnek ezzel próbálkozni új adatsorok esetén. A GINI index viszont elég gyengén teljesített. Hasonló adatsor-specifikus eredmények figyelhetőek meg a részletes eredmények között is a szintek számával, és az információnyereségi határ mértékével kapcsolatban.

Megemlítem még a CBF és a Trace adatsorokat, amelyeken a felépített modell a legtöbb esetben minimális méretű volt, azaz a 3, illetve 4 osztály mellett minimális méretű, 3 illetve 4 homogén levéllel rendelkező fát eredményeztek. Érdekes, hogy míg a Trace esetében ezzel 100%-os pontosság jelentkezett a teszthalmazon, a CBF esetén csak 89,22%. Ennek az az oka, hogy amíg a Trace 100 tanítópontján megtanult modell érvényes a maradék 100-ra, a CBF-nél a 30 idősoron tanult modell nem érvényes mind a 900 tesztpontra. Ez az adatok ismeretében várható volt, mert nagy szerencse kéne ahhoz, hogy a 30 pont minden szabályt tartalmazzon, amit a 900 idősor tartalmaz.

4.2.3.1. Eredmények a benchmark adatsorokon

A legjobb eredményeket összehasonlítottam más osztályozó módszerek pontosságával. Ezek a módszerek: Knn, Bayes-háló, C4.5, MLP, Random Forest, LMT és SVM algoritmusok, amikről az 1.3. pontban írtam.

Ezekkel az algoritmusokkal egyből két eredmény is a rendelkezésemre állt. Az első, a Weka [14] nevű programmal készített eredmények találhatóak, ahol nem történt paraméteroptimalizálás [15]. Ez azért jó összehasonlítási alap, mert az operátorok szintjén én se végeztem optimalizálást, valamint tényleg csak a legegyszerűbb operátorokat definiáltam. De emellett kíváncsi voltam, hogy az optimalizált paraméterű módszerekkel szemben hogy teljesít a módszer, ezért ugyanezen algoritmusok eredményeit, de már optimalizált paraméterekkel a DMLab oldaláról [16] is megszereztem.

Mivel az összehasonlításnál az volt a célom, hogy elhelyezzem, hogy nagyjából hol helyezkedik el a ShiftTree pontossága, mindkét eredményhalmazból származtattam az algoritmusok pontosságának maximumát, átlagát és minimumát. Az eredmény a 4.3. táblázatban látható. A bal oldali oszlopban az optimalizálatlannal, a jobboldaliban az optimalizálttal hasonlítom össze a ShiftTree eredményeit. A zöld szín azt jelenti, hogy a ShiftTree jobb a maximumnál, a sárga, hogy jobb az átlagnál, a narancs, hogy jobb a minimumnál, a piros, hogy rosszabb a minimumnál.

Adatsor	Shift Tree	Optimalizá-latlan max	Optimalizá-latlan átlag	Optimalizá-latlan min	Optimali-zált max	Optimali-zált átlag	Optimali-zált min	Shift Tree
50Words	41,32%	58,24%	42,07%	33,63%	66,37%	57,93%	41,76%	41,32%
Adiac	52,69%	56,01%	40,26%	25,06%	74,94%	59,74%	43,99%	52,69%
Beef	56,67%	50,00%	37,62%	20,00%	80,00%	62,38%	50,00%	56,67%
CBF	89,22%	32,67%	17,78%	10,33%	89,67%	82,22%	67,33%	89,22%
Coffee	67,86%	42,86%	18,88%	0,00%	100,00%	81,12%	57,14%	67,86%
ECG200	85,00%	28,00%	19,14%	11,00%	89,00%	80,86%	72,00%	85,00%
FaceAll	65,44%	44,97%	30,89%	17,57%	82,43%	69,11%	55,03%	65,44%
FaceFour	71,59%	28,41%	17,86%	11,36%	88,64%	82,14%	71,59%	71,59%
FISH	66,86%	40,00%	23,51%	14,86%	85,14%	76,49%	60,00%	66,86%
Gun Point	79,33%	22,67%	15,72%	6,67%	93,33%	84,28%	77,33%	79,33%
Lighting2	77,05%	37,70%	28,81%	19,67%	80,33%	71,19%	62,30%	77,05%
Lighting7	63,01%	45,21%	37,38%	28,77%	71,23%	62,62%	54,79%	63,01%
OliveOil	80,00%	26,67%	18,57%	13,33%	86,67%	81,43%	73,33%	80,00%
OSULeaf	56,20%	63,22%	56,02%	45,45%	54,55%	43,98%	36,78%	56,20%
Swedish Leaf	70,40%	34,40%	19,75%	13,44%	86,56%	80,25%	65,60%	70,40%
synthetic control	93,67%	19,00%	10,48%	4,00%	96,00%	89,52%	81,00%	93,67%
Trace	100%	27%	22,43%	18%	82%	77,57%	73%	100%
Two Patterns	94,18%	54,33%	24,43%	9,40%	90,60%	75,57%	45,68%	94,18%
wafer	98,25%	29,17%	5,99%	0,60%	99,40%	94,01%	70,83%	98,25%
yoga	76,03%	45,77%	29,32%	16,70%	83,30%	70,68%	54,23%	76,03%

4.3. táblázat: ShiftTree pontosságának elhelyezése

Az eredmények úgy értékelhetőek, hogy az optimalizálatlan algoritmusoknál jobb az optimalizálatlan ShiftTree, az optimalizált algoritmusok között az optimalizálatlan ShiftTree valahol a középmezőnyben van.

Érdekes megfigyelni, hogy a legrosszabb helyezést mindig az 50Words adatsoron éri el. Ennek oka az, hogy az az adatsor 50 osztályt tartalmaz és viszonylag kevés tanítópontot, a döntési fák – ami az algoritmus alapját alkotja – pedig tanulás-intenzív módszerek, azaz osztályonként sok tanítópontra van szükségük.

4.2.3.2. Eredmények a Ford adatsorokon

A Ford adatsorokon nem azt a módszert alkalmaztam, mint a korábbiakban, hanem az eredményeimet a versenyen mások által elért eredményekkel [17] hasonlítottam össze. A versenyen a pontosság mellett fontos szempont volt, hogy a hamis pozitív pontok arány, azaz a tévesen jónak minősített idősorok és hibás osztályba tartozó pontok számának aránya, minimális legyen.

Nem lettek volna viszont összehasonlíthatóak az eredmények, ha én a versenyben résztvevő halmazra (ezt nevezem én validációs halmaznak) maximalizáltam volna a modellemet, mivel az ebben megtalálható idősorok osztályai csak az eredményhirdetés után lettek nyilvánosak. Tehát azt csináltam, hogy az általam teszthalmaznak hívott halmazon a legjobb eredményt elért modellemet futtattam a validációs halmazon, mivel ha részt vettem volna a versenyben, ezt adtam volna be. A legjobb eredmény a FordA adatsor esetén a DivFunc(-2/3) decider mellett 15, vagy korlátlan mélységhatárú fán legalább 0 információnyereség határral értem el. FordB esetén a DivFunc(-1/2), 9 mélységhatárú fa, információnyereség határ nélküli beállítás bizonyult a legjobbnak.

FordA			**FordB**			**Összesített**	
Helyezés	*Pontosság*	*Hamis pozitív*	*Helyezés*	*Pontosság*	*Hamis pozitív*	*Helyezés*	*Pontosság*
1.	100,0%	0,0%	1.	86,2%	12,7%	1.	92,2%
2.	99,6%	0,0%	2.	84,3%	23,7%	2.	91,4%
3.	97,8%	1,8%	3.	83,8%	25,9%	3.	91,3%
4.	96,7%	2,1%	4.	83,2%	23,4%	4.	90,9%
5.	96,5%	2,9%	5.	83,2%	27,4%	5.	90,8%
6.	96,0%	3,7%	6.	82,8%	27,2%	6.	90,1%
7.	95,5%	4,6%	7.	82,5%	24,9%	7.	89,8%
8.	95,4%	5,0%	8.	79,7%	33,7%	8.	89,2%
9.	94,9%	3,5%	9.	78,9%	27,4%	9.	88,7%
10.	94,5%	6,6%	10.	68,8%	50,4%	10.	87,9%
11.	93,9%	6,8%	11.	68,1%	35,9%	11.	84,3%
12.	92,5%	7,6%	12.	67,7%	45,9%	12.	80,9%
13.	89,6%	9,5%	**ShiftTree**	**67,3% (75,2%)**	**23,5% (18,6%)**	13.	79,3%
14.	85,6%	17,2%	13.	66,7%	34,4%	14.	77,6%
15.	83,8%	15,0%	14.	64,6%	68,8%	15.	76,1%
16.	81,6%	27,3%	15.	63,6%	75,6%	**ShiftTree**	**76,0% (81,0%)**
ShiftTree	**81,4% (84,6%)**	**18,2% (14,8%)**	16.	62,3%	47,3%	16.	74,1%
17.	80,7%	39,0%	17.	61,9%	75,8%	17.	74,0%
18.	76,9%	25,8%	18.	59,3%	72,3%	18.	73,4%
19.	75,3%	40,2%	19.	54,9%	49,1%	19.	72,8%
20.	65,9%	24,2%	20.	51,2%	48,1%	20.	61,7%

4.4. táblázat: ShiftTree elhelyezése a Ford verseny eredményei között

A 4.4. táblázatban látható, hogy a versenyen a 20 legjobb pontosságot elért megoldás között hol helyezkedik el a ShiftTree eredménye (sárgával kiemelve). A zárójelben szereplő szám a teszthalmazon elért eredményeket jelenti. Ez nem számít bele a versenybe, csak tájékoztatás végett közlöm. Mint látható, a zajos adaton valamennyivel jobb a többi megoldással összehasonlítva, mint a preparált adatokon. Az is megfigyelhető, hogy a zajos adatokon kiemelkedően alacsony hamis pozitív hibával sikerült megoldania a feladatot a többi megoldáshoz képest. Megint hangsúlyoznám, hogy az algoritmuson nem végeztem paraméteroptimalizálást, tehát az elért helyezések kifejezetten jónak számítanak.

4.3. A felépített modellek értelmezhetősége

A gyakorlati problémák megoldása során általában fontos, hogy a felépített modellek érthetőek, megmagyarázhatóak legyenek. Sokszor ez fontosabb, mint maga a pontosság, hiszen az emberek nem szívesen bíznak meg olyan módszerekben, amik csak az eredményt közlik a magyarázatot viszont nem (pl. neurális hálók), hiszen az érthető modellek elemzése közben észrevehető, ha a módszer nem azt tanulta meg, amit szerettünk volna. A döntési fák minden esetben megindokolják, hogy egy adott rekordot miért soroltak egy adott osztályba, és mivel a ShiftTree a döntési fákra épült, minden felépített modell tartalmazza a modell magyarázatát is. A magyarázat viszont a döntési fák esetében is csak viszonylag kis modellek esetén átlátható, nincs ez másként a ShiftTree-nél sem. Éppen ezért két viszonylag kisméretű, kompakt modellen mutatom be, hogy az algoritmus miként magyarázza meg az osztályozást.

A futtatáshoz használt ESO-k és CBO-k megegyeznek a 4.2.1. pont alatt leírtakkal, a vágások jósági értékét entrópia alapú decider-rel számolom.

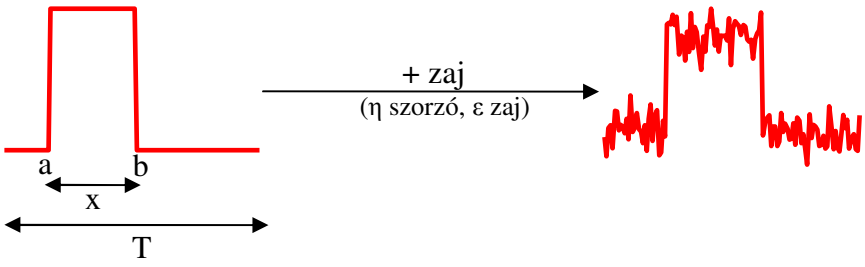
4.3.1. CBF-modell értelmezése

A CBF egyike azoknak az adatsoroknak, amelyekhez kicsi, kompakt modell épült. Ráadásul szinte az egyetlen az adatsorok közül, ami emberek számára is könnyen értelmezhető idősorokat tartalmaz. Ezért választottam ennek a modelljét elemzésre.

4.3.1.1. A CBF adatsor

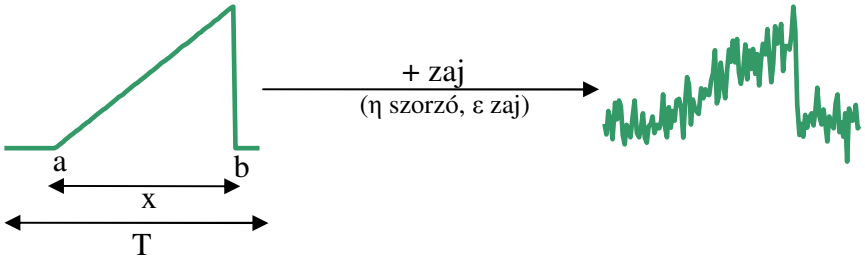
A CBF adatsor az alábbi három osztályt tartalmazza [18]:

Cylinder – x hosszú négyszögjel



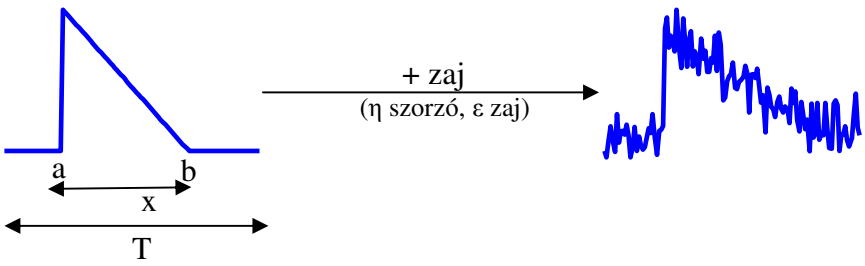
$$c(t) = (6 + \eta) * X_{[a,b]}(t) + \varepsilon(t)$$

Bell – x hosszú háromszögjel



$$b(t) = (6 + \eta) * X_{[a,b]}(t) \frac{(t - a)}{(b - a)} + \varepsilon(t)$$

Funnel – x hosszú fűrészfogjel



$$f(t) = (6 + \eta) * X_{[a,b]}(t) \frac{(b - a)}{(b - t)} + \varepsilon(t)$$

Az egyes változók jelentései:

- a – az intervallum kezdete, ahol a jel kezdődik
- b – az intervallum vége, ahol a jel véget ér
- x – a jel hossza (x = b – a)
- T – az idősor teljes hossza
- $X_{[a,b]}$ – az [a,b] intervallumon 1, azon kívül 0 (a matematikai leíráshoz kell)
- η, ε – zajkomponensek standard normális eloszlás szerint

Változók értékei az adatsor idősoraiban:

- Az idősorok hossza: fixen T = 128
- A jel hossza: $32 \leq x \leq 96$
- A jel kezdete: $16 \leq a \leq 32$
- A fentiekből következően a jel vége: $48 \leq b \leq 128$

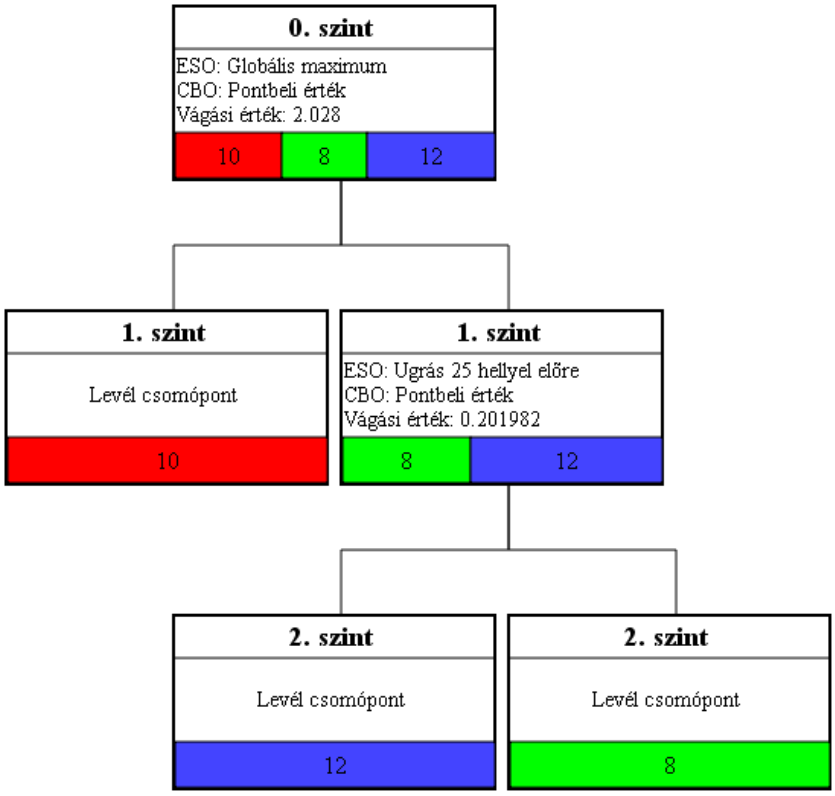
Az adatsor osztályeloszlása

- Tanítóhalmaz – C: 10, B: 8, F: 12, összesen 30 idősor
- Teszthalmaz – C: 300, B: 298, F: 302, összesen 900 idősor

Egyéb jellemzők:

- Az adatok z normalizáltak

4.3.1.2. A felépített modell



4.1. ábra: A CBF modell

A 4.1. ábra mutatja a CBF adatsor tanítóhalmaza alapján felépített modellt. A modell leírása alatt minden csomópontban jelzi az oda került tanítóminták számát, különböző színnel jelölve a különböző osztályokhoz tartozó idősorokat (Piros – C; Zöld – B; Kék – F). Mint látható, a modell minimális, azaz az algoritmus a három osztályhoz három homogén levéllel rendelkező fát épít. Ez a minimális méret jól értelmezhetővé teszi a modellt.

Az osztályozás hatékonysága:
A találati arány 89,22%, de érdemes megnézni, hogy helytelen besorolás esetén melyik jelet melyikkel téveszti össze, ez látható a 4.5. táblázatban.

		Besorolt osztály		
		C	F	B
Tényleges osztály	C	278	7	15
	F	7	245	50
	B	9	9	280

4.5. táblázat: A CBF osztályozási mátrixa

4.3.1.3. A modell elemzése

A modell értelmezését két részre bontottam vágásonként: előbb az 1. szinten lévő egyszerűbb vágást, a Funnel és Bell jelek szétválasztását magyarázom meg. Ezután következik a Cylinder és a háromszögjelek szétválasztásának, azaz a 0. szinten történt vágásnak a magyarázata, mivel ez utóbbi nehezebben megfejthető.

4.3.1.3.1. Funnel-Bell szétválasztás

A modell egyszerűbben értelmezhető része a Funnel és a Bell jelek szétválasztása. Ez az 1. szinten történik, a 0. szint ESO-jának beállítása után, tehát kiindulásként az adott idősor maximumánál van a szem.

A Bell maximuma a jel vége felé van (zaj nélküli esetben pontosan b-nél). Ha onnan ugrunk még 25-öt, akkor zaj nélküli esetben a [73,128] intervallumon vagyunk valahol, ahol az érték a minimum. Zaj esetén az intervallum alja valamivel alacsonyabb, és az érték a minimum+zaj adott pontbeli értékével egyezik meg.

A Funnel maximuma a jel elejének környékén van (zaj nélküli esetben pontosan a-nál). Ha onnan ugrunk még 25-öt, akkor még mindig az [a,b] intervallumon belül vagyunk, mivel annak minimális hossza 32. Zajos esetben viszont, ha a maximum a-tól több, mint 7 távolságra van, akkor a 25-ös ugrással túlugrunk a jelen, és az érték itt is a minimum+zaj adott pontbeli értéke lesz. Ebben az esetben tévesztjük össze a Funnel-t a Bell-lel (Funnel helyett Bell-t mondunk). A táblázatban is látszik, hogy itt történik a legtöbb (50) összetévesztés.

Ha nem ugrunk túl a jelen, akkor viszont egyértelmű, hogy a Funnel pontbeli értéke (a zajt leszámítva) nagyobb, mint a Bell-é, tehát a modell alapvetően helyes. A zaj beleszólhat itt is, ami által a Bell-t Funnel-nek kiáltjuk ki (a zaj miatt magasabb a pontbeli értéke), de látszik, hogy ez ritka esemény (9 a táblázatban). (Ugyanígy a Funnel-Bell összetévesztés oka is lehet az, hogy a zaj miatt a Funnel pontbeli értéke kisebb, mint a meghatározott határ, de ez kisebb valószínűségű, mint az, hogy túlugrottunk a jelen, és ezért tévesztettük össze őket. Még egy összetévesztési lehetőség az a ritka eset, ha a Bell az intervallum végéig kiér, azaz b=128. Ekkor a Bell-t Funnel-nak mondjuk, de ez messze a legritkább összetévesztési ok.)

4.3.1.3.2. Négyszögjel és háromszögjel szétválasztás

A 0. szint vágásának magyarázata összetettebb. Ha nem vesszük figyelembe a z normalizálást, akkor semmi értelme a globális maximumot összehasonlítani, mert az a 4.3.1.1. pont elején leírtak szerint minden osztály esetén 1 (+zaj). Viszont ha belegondolunk, hogy az adatsor z normalizált, akkor megtalálhatjuk a vágás okát.

A továbbiakban tekintsük a zajmentes esetet:

A négyszögjeles idősor átlaga: $\frac{xy}{T}$, ahol y a maximális érték (azaz 1).

A négyszögjeles idősor szórása: $\sqrt{\frac{x\left(\frac{xy}{T}-y\right)^2+(T-x)\left(\frac{xy}{T}-0\right)^2}{T}}=\sqrt{\frac{y^2\left(x-\frac{x^2}{T}\right)}{T}}=y\sqrt{\left(\frac{x}{T}-\frac{x^2}{T^2}\right)}$

A z normalizált négyszögjeles idősor maximuma tehát:

$$y_{négyszög-\max}=\frac{y-\frac{xy}{T}}{y\sqrt{\left(\frac{x}{T}-\frac{x^2}{T^2}\right)}}=\frac{\sqrt{\left(1-\frac{x}{T}\right)^2}}{\sqrt{\frac{x}{T}\left(1-\frac{x}{T}\right)}}=\sqrt{\left(1-\frac{x}{T}\right)\frac{T}{x}}=\sqrt{\frac{T}{x}-1}$$

Ami T értékét és x minimális és maximális értékét behelyettesítve:

$$y_{négyszög-\max\,MAX}=\sqrt{3}=1{,}732$$

$$y_{négyszög-\max\,MIN}=\sqrt{\frac{4}{3}}=1{,}155$$

A két háromszögjel statisztikai jellemzői ugyanazok, hiszen a két háromszögjel tükörképe egymásnak.

A háromszögjeles idősor átlaga: $\frac{\sum_{i=0}^{x-1} y-\frac{y}{x}i}{T}=\frac{xy-\frac{(x-1)xy}{2x}}{T}=\frac{y(x-1)}{2T}$

A háromszögjeles idősor szórása: $\sigma^2=\frac{(T-x)\left(\frac{y(x-1)}{2T}-0\right)^2+\sum_{i=0}^{x-1}\left(\frac{y(x-1)}{2T}-y+\frac{y}{x}i\right)^2}{T}$

$$T\sigma^2=(T-x)\left(\frac{y(x-1)}{2T}\right)^2+x\left(\frac{y(x-1)}{2T}-y\right)^2+2\frac{(x-1)xy}{2x}\left(\frac{(x-1)y}{2T}-y\right)+\frac{y^2}{x^2}\frac{(x-1)x(2x-1)}{6}$$

$$T\sigma^2=T\left(\frac{y(x-1)}{2T}\right)^2-2x\left(\frac{y^2(x-1)}{2T}\right)+xy^2+\frac{((x-1)y)^2}{2T}-y^2(x-1)+\frac{y^2}{x}\frac{(x-1)(2x-1)}{6}$$

$$T\sigma^2=\frac{3((x-1)y)^2}{4T}-2x\left(\frac{y^2(x-1)}{2T}\right)+y^2+\frac{y^2}{x}\frac{(x-1)(2x-1)}{6}$$

$$T\sigma^2=y^2\left(\frac{(x-1)(3x-3-4x)}{4T}+\frac{(x+1)(2x+1)}{6x}\right)=y^2\left(\frac{(x+1)(2x+1)}{6x}-\frac{(x-1)(x+3)}{4T}\right)$$

$$\sigma=\sqrt{\frac{y^2}{T}\left(\frac{(x+1)(2x+1)}{6x}-\frac{(x-1)(x+3)}{4T}\right)}$$

Ezekből a z normalizált háromszögjeles idősor maximuma:

$$y_{háromszög-\max}=\frac{y-\frac{y(x-1)}{2T}}{\sqrt{\frac{y^2}{T}\left(\frac{(x+1)(2x+1)}{6x}-\frac{(x-1)(x+3)}{4T}\right)}}=\sqrt{\frac{\left(1-\frac{(x-1)}{2T}\right)^2}{\frac{1}{T}\left(\frac{(x+1)(2x+1)}{6x}-\frac{(x-1)(x+3)}{4T}\right)}}$$

Egyszerűsítve: $y_{háromszög-\max} = \sqrt{\frac{3x(2T-x+1)^2}{2t(x+1)(2x+1)-3x(x-1)(x+3)}}$

Néhány érték kipróbálása után azt vettem észre, hogy a [32,96] intervallumon ez a kifejezés monoton csökkenő (hasonlóan a négyszögjel maximumához), tehát a szélsőértékeit az intervallum széleinél veszi fel.

$y_{háromszög-\max\ MAX} = 3{,}276$

$y_{háromszög-\max\ MIN} = 1{,}869$

Innen már látszik, hogy zaj nélküli esetben a háromszögjel maximuma mindig nagyobb, mint a négyszögjelé, azaz a vágás helyes. A zaj hozzáadásával a szórások csökkenhetnek és nőhetnek is, ezért a maximumok intervallumai átfedhetik egymást. A fenti táblázatban látható, hogy viszonylag kevés négyszögjelet és háromszögjelet keverünk össze, de azért létezik a jelenség. Ennek az oka az, hogy a hozzáadott zaj miatt eltérő módon növekedhet/csökkenhet az egyes idősorok szórásai, ezért a zajos jelek maximumainak intervallumai átfedhetik egymást. Ezért fordulhat elő, hogy a vágásban szereplő 2,028 érték nagyobb, mint a háromszögek maximumának zajnélküli minimuma. A tanítóhalmazban kettő négyszögjel maximuma is átlóg a háromszögjelek maximumainak intervallumába, viszont egy háromszögjel se lóg át a négyszögjelek tartományába, ezért a meghúzott határ a zajmentes háromszögjelek maximumintervallumának alját levágja. Ez eredményezi azt, hogy több háromszögjelet tippelünk négyszögjelnek, mint amennyi négyszögjelet háromszögnek.

A teszt adathalmaz megvizsgálása során kiderült, hogy a négyszögjelek maximum tartománya a teszthalmazban $1{,}182 \le y_{z\max} \le 2{,}379$

A háromszögeké $1{,}768 \le y_{z\max} \le 3{,}793$

A megállapított határ 22 négyszögjelre mondja, hogy háromszög (7,33%-a a négyszögeknek) és 16 háromszögre mondja, hogy négyszög (2,66%-a háromszögeknek). Ez nem egy jelentős mennyiség. Ennek ellenére megvizsgáltam a teszt adathalmaz idősorainak maximumait a vágási határ környezetében, hogy lehetne-e valahogyan javítani a pontosságon.

Osztály	**Maximum**
Cylinder	*2.014*
Cylinder	*2.02178*
Bell	*2.02352*
Cylinder	*2.02536*
Cylinder	*2.02781*
Bell	*2.02793*
Cylinder	2.03044
Cylinder	2.0367
Funnel	2.03707
Bell	2.03904
Funnel	2.04296

4.6. táblázat: A 0. szintű vágás környezete a CBF modellben

A 4.6. táblázatban találhatóak a vágás környezetébe tartozó idősorok a teszthalmazból. A táblázat utolsó sora utáni részen elvétve, egyesével fordul elő a maradék 20 Cylinder jel. A táblázat első sora előtt szintén elszórva szerepel 7 Funnel és 7 Bell jel. Látszik, hogy 2 Cylinder jel helyes besorolásával lehetne javítani a pontosságot úgy, hogy a vágási határt

2.03044 fölé növeljük. Ez 0,22%-nyi, tehát elhanyagolható mennyiségű javulást eredményezne, viszont jelentős beavatkozást jelentene egy ilyen automatikus módszer esetén.

4.3.1.4. CBO-k definiálási sorrendjének hatása

Megvizsgáltam, hogy van-e valamilyen hatása annak, hogy az egyes operátorokat milyen sorrendben definiálom a programban. Azt tapasztaltam, hogy igen, jelentősen javíthat/ronthat a pontosságon a definiálási sorrend. Ennek az az oka, hogy a program akkor áll át egy adott ESO-CBO párról egy másikra, ha az nagyobb jóságértéket produkáló vágást generál a tanítóhalmazon. Ha több ESO-CBO pár is ugyanolyan jól vágja szét a tanítóhalmazt, akkor mindig a legelső lesz kiválasztva. A teszthalmazon viszont ezek a tanítás szempontjából egyformán jó modellek eltérő pontosságot eredményezhetnek.

Ha megfordítom az ESO-k definiálási sorrendjét, akkor az ESONext(25) helyett a modellben a Funnel-Bell szétválasztásnál az ESOPrev(25) szerepel. Ez fordított irányban ugyanazt csinálja, mint amit a 4.3.1.3. pontban leírtam, ezért a 2. szintű levelek megcserélődtek. Ehhez az ESO-hoz a modellben egy standard normális eloszlás szerinti súlyozó CBO-t választ a fa, azért, mert a tanítóminták elején a pontbeli értéket használó CBO a zaj miatt nem ad eléggé pontos vágást.

		Besorolt osztály		
		C	**F**	**B**
Tényleges osztály	**C**	278	11	11
	F	7	282	13
	B	9	4	285

4.7. táblázat: A CBF osztályozási mátrixa fordított ESO definiálási sorrendnél

A 4.7. táblázatban látható az ehhez a konfigurációhoz tartozó osztályozási mátrix. Az osztályozás pontossága: 93,88%. A jelentős növekedés onnan származik, hogy a zajt jobban szűrő CBO miatt a Funnel és Bell jeleket kevésbé téveszti össze a modell.

Ha az eredeti ESO sorrend mellett a CBO-kat eltérő sorrendben definiáljuk, azaz a CBONormal(0,1,0.05) operátort kisebb sorszámmal adjuk hozzá a feltételállítóhoz, mint a CBOSimple-t, akkor az előzőhöz közeli pontosságú modellt kapunk. Mint ahogy várható volt, most az ESONext(25) mellé a CBONormal-t választja az algoritmus az 1. szinten.

		Besorolt osztály		
		C	**F**	**B**
Tényleges osztály	**C**	278	8	14
	F	7	281	14
	B	9	7	282

4.8. táblázat: A CBF osztályozási mátrixa súlyozott átlagot számító CBO-t a CB elején definiálva

A 4.8. táblázatban látható az ehhez a konfigurációhoz tartozó osztályozási mátrix. Az osztályozás pontossága: 93,44%. Az eredeti pontossághoz képesti növekedést itt is a jobb zajszűrő képesség nagyarázza.

Kipróbáltam, hogy mi történik, ha a szélesebb intervallumon súlyozó CBNormal(0.5,4,0.01)-t definiálok elsőnek. Ebben az esetben ezt választotta a fa a vágásnál, és még pontosabb eredményt kaptam, mivel szélesebb intervallumon történt az átlagolás és így a zajszűrés.

		Besorolt osztály		
		C	F	B
Tényleges osztály	C	278	8	14
	F	7	292	3
	B	9	9	280

4.9. táblázat: A CBF osztályozási mátrixa széles súlyozott átlagot számító CBO-t a CB elején

Az eredmény a 4.9. táblázatban látható, a nagyobb zajszűrés hatására a pontosság: 94,44%.

Végül a fordított ESO sorrend és fordított CBO sorrend tesztje következett. Ebben az esetben az ESOPrev(25) és a szélesebb intervallumon súlyozó normális eloszlással operáló CBO került a modell 1. szintjére.

		Besorolt osztály		
		C	F	B
Tényleges osztály	C	278	10	12
	F	7	289	6
	B	9	4	285

4.10. táblázat: CBF osztályozási mátrixa széles átlagoló CBO-val a CB elején, fordított ESO sorral

Az eredmény a 4.10. táblázatban látható. A jel elején történő vizsgálat egy kicsivel pontosabb eredményt ad, mint a jel végi vizsgálat. A pontosság: 94,66%.

A fentiekből is látszik, hogy ha kis tanítóhalmazon több CBO, illetve ESO ugyanazt a modellt eredményezi, akkor jelenleg a kisebb sorszámmal definiált lesz kiválasztva. Mint látható, ez nem feltétlenül a legjobb eredményt adja a tesztadatokon. Ugyanakkor ez nem róható fel a módszernek, mert a tanítóhalmazon nem tud különbséget tenni a modellek között. Megoldás egy olyan automatizmus lehet, ami ciklikusan permutálja a CBO-kat (és esetleg az ESO-kat), és minden konfigurációban épít egy modellt. Ha ez a modell ugyanolyan jó, mint az eredeti, akkor hozzáadja az elkészült modellekhez. Ha jobb, akkor megtartja, és törli azokat, amiket eddig tárolt. Rosszabb nem lehet, mert akkor automatikusan másik CBO-t választana a fa, és így az új futás eredménye pontosan megegyezne egy korábbi modellel. A módszer hátránya, hogy a futási idő jelentősen megnő, mert ha az ESO-kra és a CBO-kra is engedélyezzük, akkor $N_{ESO}*N_{CBO}$ modellt épít, ami ugyanennyiszeresére növeli a futási időt. Takarékosabb megoldás csak a CBO-kat permutálni, mivel ezek általában nagyobb hatással lehetnek az eredményre, mint az ESO-k. Mivel a CBO-k nincsenek olyan sokan, ez a futási idő többlet a modell megnövekedett pontosságért cserébe elfogadható lehet.

Ha nem akarjuk a fenti módszert használni, akkor figyeljünk arra, hogy a szélesebb intervallumon súlyozó CBO-kat definiáljuk a feltételállító elején. Mivel ha két CBO a tanítópontokat ugyanúgy vágja, akkor érdemes a szélesebbet használni, mert annak nagyobb zajszűrő hatása van. Ennek ellenére a továbbiakban is az eddig használt sorrendet használom, hogy az eredmények összehasonlíthatóak legyenek.

4.3.2. A Trace-modell értelmezése

A Trace több szempontból is különleges adatsor. Egyrészt sok beállítás mellett minimális modellt ad, mint a CBF. Viszont attól eltérően a pontossága a legtöbb esetben 100%. Az is érdekes, hogy fordított tanulásnál, azaz akkor, amikor a teszthalmazon tanítom a modellt, és a tanítóhalmazon tesztelem, lényegében ugyanazt a modellt adja, mint a normális tanulásnál. Eltérés csak a vágási értékek között figyelhető meg minimálisan.

Emiatt a sok érdekes tulajdonság miatt választottam ezt az adatsort a második modell elemzéshez.

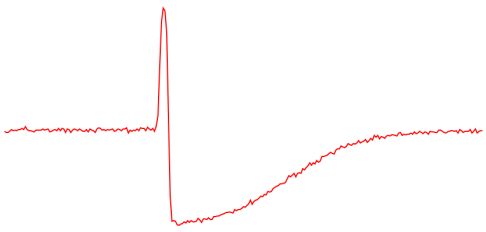
4.3.2.1. A Trace adatsor

A Trace adatsor [19] olyan mesterséges idősorokat tartalmaz, amik különböző tranzienseket írnak le, amik nukleáris erőművekben előfordulhatnak. Az eredeti adatsor 16 osztályt tartalmaz, minden osztályhoz 50 idősor tartozik, és minden idősornak 4 változója van. (Úgy áll elő a 16 osztály, hogy a 4 típusú változó mindegyike kétféle idősor lehet, és ezeket az összes lehetséges módon kombinálva állnak elő többváltozós idősorok. (2^4))

A rendelkezésemre álló adatsor csak 4 osztályt tartalmaz, mégpedig úgy, hogy az eredeti adatsorból a második és hatodik osztály 2. változóját, valamint a harmadik és hetedik osztály 3. változóját [20] tartalmazza.

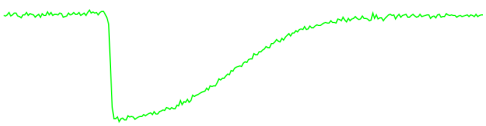
A négy osztály egy-egy példánya a következő módon néz ki:

Trace1:



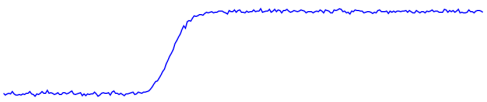
Egy ideig nagyjából állandó érték, majd egy magas tüske következik. A tüske után gyorsan leesik az érték, majd viszonylag lassan visszaáll az eredeti érték környékére (valamivel az eredeti érték alá).

Trace2:



A tüskét leszámítva ugyanaz, mint az előző. Egy ideig nagyjából állandó, majd hirtelen leesik. Utána viszonylag lassan áll vissza az eredeti érték környékére (itt is valamennyivel alá).

Trace3:



Az előzőektől eltérő idősor. Egy adott értéken áll egy ideig, majd közepes sebességgel emelkedik egy másik értékig.

Trace4:

Az előzőhöz nagyon hasonló, csak az emelkedés utáni állandósult szakaszon egy kisebb ingadozás is megfigyelhető.

Az 1-es és 2-es osztályok nagyjából ugyanakkora értékről indulnak. Ez az érték megfelel annak, ami a 3-as és 4-es osztályok végén jellemző. A 3-as és 4-es osztályok kb. ugyanakkora értékről indulnak. Ez az érték már a negatív tartományba esik. Az 1-es és 2-es osztályok minimuma szintén a negatív tartományban van és kisebb, mint a 3-as és 4-es osztályok indulási értékei.

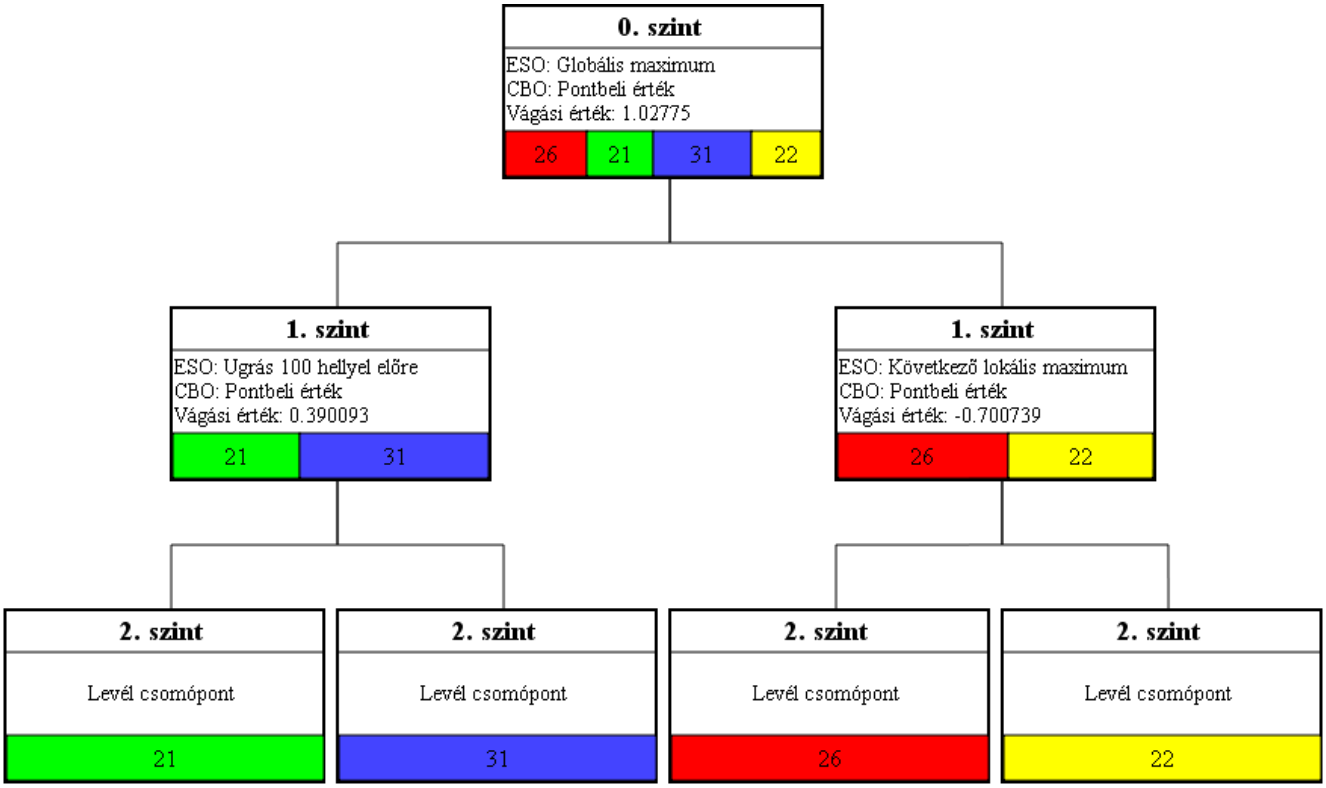
Az adatsor osztályeloszlása

- Tanítóhalmaz
  - Trace1: 26, Trace2: 21, Trace3: 31, Trace4: 22
  - Összesen 100 idősor
- Teszthalmaz
  - Trace1: 24, Trace2: 29, Trace3: 19, Trace4: 28
  - Összesen 100 idősor
- Összesen 50 minta minden osztályból

Egyéb jellemzők

- Az idősorok hossza 275.
- Az adatok z normalizáltak.
- Nincs multiplikatív zaj.
- Additív zaj van.

4.3.2.2. A felépített modell



4.2. ábra: A Trace modell

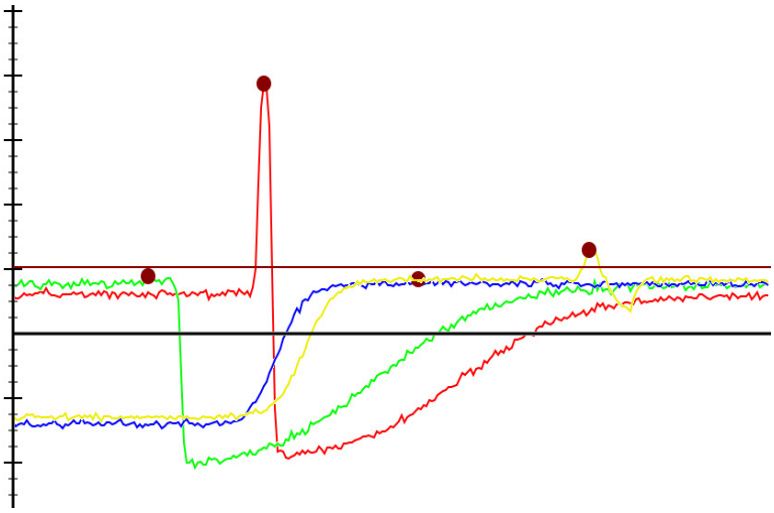
A 4.2. ábra mutatja a Trace adatsor tanítóhalmaza alapján felépített modellt. A modell leírása mellet minden csomópontban jelzi az oda került tanítóminták számát, különböző színnel jelölve a különböző osztályokhoz tartozó idősorokat.

(Piros – Trace1; Zöld – Trace2; Kék – Trace3; Sárga – Trace4).

Mint látható, a modell minimális, azaz a négy osztályhoz az algoritmus egy négy homogén levéllel rendelkező fát épít. Ráadásul egyből látható, hogy ez a fa minimális szintű. Azaz a modell nagyon könnyen meg tud különböztetni két osztályt a másik kettőtől, majd ezeket a párokat vágja szét nagyon hatékonyan.

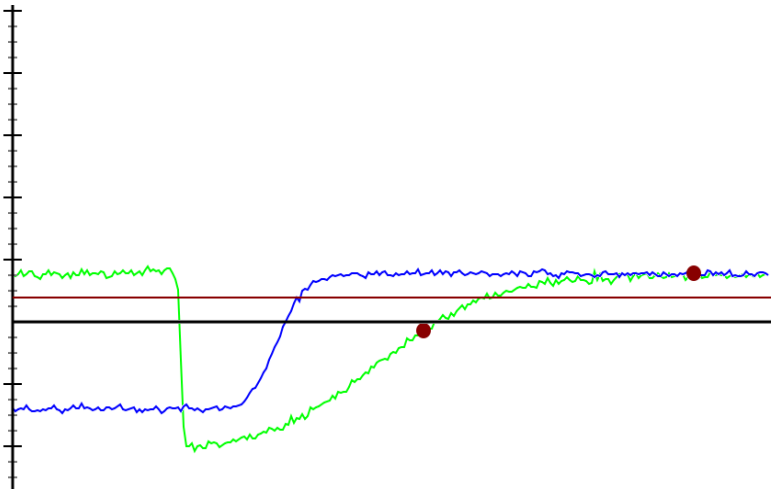
4.3.2.3. A modell elemzése

A 0. szintű vágás (4.3. ábra) a globális maximumot használja. Azt használja ki a modell, hogy ezek az adatok tranziensek, azaz az állandósult értékek nagyjából azonosak és, hogy az adatsoron nem figyelhető meg multiplikatív zaj. Az értékekhez képest kis additív zaj viszont nem elég ahhoz, hogy annyira módosítsa a Trace1 és Trace4 maximumát, hogy azok összetéveszthetőek legyenek a konstans szakaszok értékekeivel, ahol a Trace2 és Trace3 osztályok maximumai találhatóak.



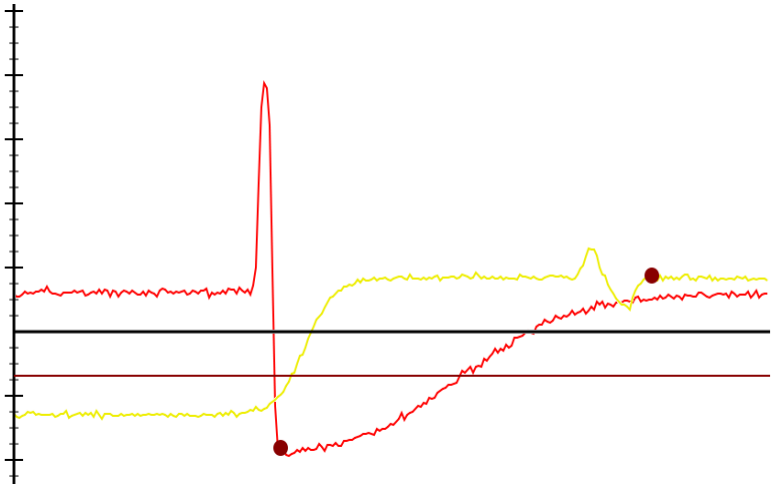
4.3. ábra: A Trace modell 0. szintű vágása

A Trace2 és Trace3 közötti vágás (4.4. ábra) alapja az, hogy a Trace3 maximuma az állandósult szakaszon, a jel végén van, míg a Trace2-nél ez a tranziens előtti szakaszon van. Innen 100 ugrás a Trace3-nál az állandósult szakaszon belül marad (esetleg az idősor végére kerül), a Trace2 esetében viszont nagy valószínűséggel a tranziensen belülre érkezünk (annak is a végére). Mivel a Trace2 esetében a tranziens vége lassan konvergál az eredeti értékhez (e^{-x} szerű, ami által az idősor végén a kezdeti értéknél kisebb értékű szakaszt látunk közel állandónak), ezért a Trace2 esetén a maximumtól 100-ra lévő érték kisebb lesz, mint a Trace3 esetén az állandósult érték (ami megegyezik a Trace2 kezdeti értékével).



4.4. ábra: A Trace2 és Trace3 osztályok megkülönböztetése

A Trace1 és Trace4 közötti vágás (4.5. ábra) azért jöhet létre, mert nincs multiplikatív zaj. Ennek köszönhetően a Trace1 tüskéje utáni meredek esésben nincs lokális maximum. A következő ilyen valahol a negatív tartományban fordul elő az additív zaj miatt (ha nem lenne zaj, akkor az idősor végén lenne a következő lokális maximum). Trace4 esetén maximum az állandósult értékben bekövetkező ingadozás maximuma. A következő lokális maximum nagy valószínűség szerint az állandósult értéken van (kis valószínűséggel lehet az ingadozás belsején), de mindenképpen a pozitív tartományban található.



4.5. ábra: A Trace1 és Trace4 osztályok megkülönböztetése

4.4. Keresztvalidációs mérések

A keresztvalidációs mérések esetemben azt a célt szolgálják, hogy az algoritmus mélyebben rejtőző tulajdonságait megvizsgáljam.

Ezen mérések során eltérek a hivatalosan kiadott tanító- és teszthalmazoktól, és mindkét halmazt összekeverve, majd adott arányban felbontva új tanító- és tesztadathalmazokat hozok létre. Ezáltal lehetőségem nyílik azt megvizsgálni, hogy a futási idő nagysága és a pontosság hogyan függ össze a gyakorlatban a tanítópontok számával az egyes adatsorok esetében. Valamint, azt is megvizsgálom, hogy miként sorolhatóak kategóriákba az egyes adatsorok a futási idő- vagy a pontosságfüggvényük alapján.

4.4.1. Mérési módszer

Ebben a pontban definiálom, hogy pontosan mit értek keresztvalidáció alatt, ismertetem a pontosság és a futási idő mérésére alkalmazott módszereket, és leírom a tesztkörnyezetet, valamint a program konfigurációját.

4.4.1.1. Keresztvalidáció

Mind a 22 adatsoron a következőket végeztem el:

- TEST adatsor beolvasása egy megfelelő struktúrába
- TRAIN adatsor beolvasása és hozzáadása a struktúrához
- A struktúra idősorainak random szerű felcserélése a következő módon:
    - 0 és a struktúrában aktuálisan lévő idősorok száma közötti véletlen szám sorsolása
    - A sorsolt számú idősor kivétele a struktúrából és hozzáadása az eredménystruktúrához
    - Mindezt addig folytatva, amíg az eredeti struktúra el nem fogyott
- A randomizált sorrendű tanítópontok (idősorok) első X százalékán tanítottam egy ShitTree-t, megmértem a futási időt és ellenőriztem a modell pontosságát a tanításban részt nem vevő 100-X százaléknyi idősoron.
- A randomizált sorrendű tanítópontok (idősorok) 5%-ától kezdve újra X százalékon tanítva egy új modellt készítettem, a mérendő adatokat itt is feljegyeztem (a pontosság ellenőrzése a 0% - 5% és 5+X% – 100% intervallumok idősorain).
- Az előzőekhez hasonlóan méréseket végeztem a tanítópontok 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95 százalékától kezdve a teljes minta X százalékát tanítópontként használva.
- Ha X + a kezdeti százalék nagyobb, mint 100, akkor az átlógó százalékértékeknél mod100-zal számolok.
- X értékei: 5, 10, 15, …, 95.

Azaz adatsoronként 19*20 = 380 modellépítés történik (összesen 22*380 = 8360). Adatsoronként az idősorok egy adott százalékával 20 mérés készül, ezek alapján már viszonylag pontos képet kapok egy adott mennyiségű idősorral való tanítás pontosságáról, és idejéről. A 20 mérés eredményét Excelben átlagoltam, valamint kiszámoltam a mérések szórását is.

4.4.1.2. A pontosság mérése

Az adott adatsoron a tanításban részt nem vevő idősorokon ellenőrzöm, hogy a felépített modell az osztályok hány százalékát találja el helyesen. Pontosság = (helyesen eltalált osztályok száma) / (a tanításban részt nem vevő idősorok száma).

A módszer hátránya, hogy a kisméretű (kevés idősort tartalmazó) adatsorok esetén a 80+ százalékos tanítások esetén az ellenőrző idősorok száma nagyon kevés, ezért 1-1 hiba sokkal jobban megmutatkozik a pontosság tekintetében (Pl.: a Beef adatsor 60 idősort tartalmaz, 95%-os tanításnál 3 ponton tesztelünk). Ezért ott a szórás is megnő, az eredmények eléggé randomszerűek lesznek. Erre figyelni kell az eredmények értékelésénél.

Az egyszerű találati arány mellett elmentettem a találati mátrixot is, hiszen sokszor nem csak a pontosság értéke számít, hanem egyes típusú hibákat (pl. hamis pozitív) minimalizálni szeretnénk.

4.4.1.3. A futási idő mérése

Mivel az egyes adatsorok idősorainak száma/hossza jelentősen eltér, és viszonylag kis minták futásidejére is pontosan van szükség, ezért nagy pontosságú időzítőt használok. A QualityPerformanceCounter(LARGE_INT* li) függvény a meghívásakor lekéri a számítógép bekapcsolása óta megtörtént TICK-ek számát. A TICK-ek száma másodpercenként a tesztgépen: 3579545. A nagypontosságú időzítő átfordulásával nem kell számolnunk, mivel 2^{64} TICK a tesztgépen $5{,}15*10^{12}$ másodpercig tart, ami egy kicsit több mint 163ezer év.

4.4.2. A program konfigurációja

Decider: DEntropy
ESO-k (sorrendben): NextMax, NextMin, Next(1), Next(5), Next(10), Next(25), Next(50), Next(100), Next(200), Next(400), Max, Min, Prev(1), Prev(5), Prev(10), Prev(25), Prev(50), Prev(100), Prev(200), Prev(400), PrevMax, PrevMin
CBO-k (sorrendben): Simple, Normal(0, 1, 0.05), Exp(1, 0.05), Normal(0, 0.5, 0.01), Exp(0.5, 0.01), Normal(0.5, 4, 0.01), Exp(2, 0.01), AVG(5)
Információnyereség minimális határa: nincs
Szintkorlát: nincs

4.4.3. Tesztelési környezet

A laptopomat használtam tesztgépnek, a fontosabb paraméterek:

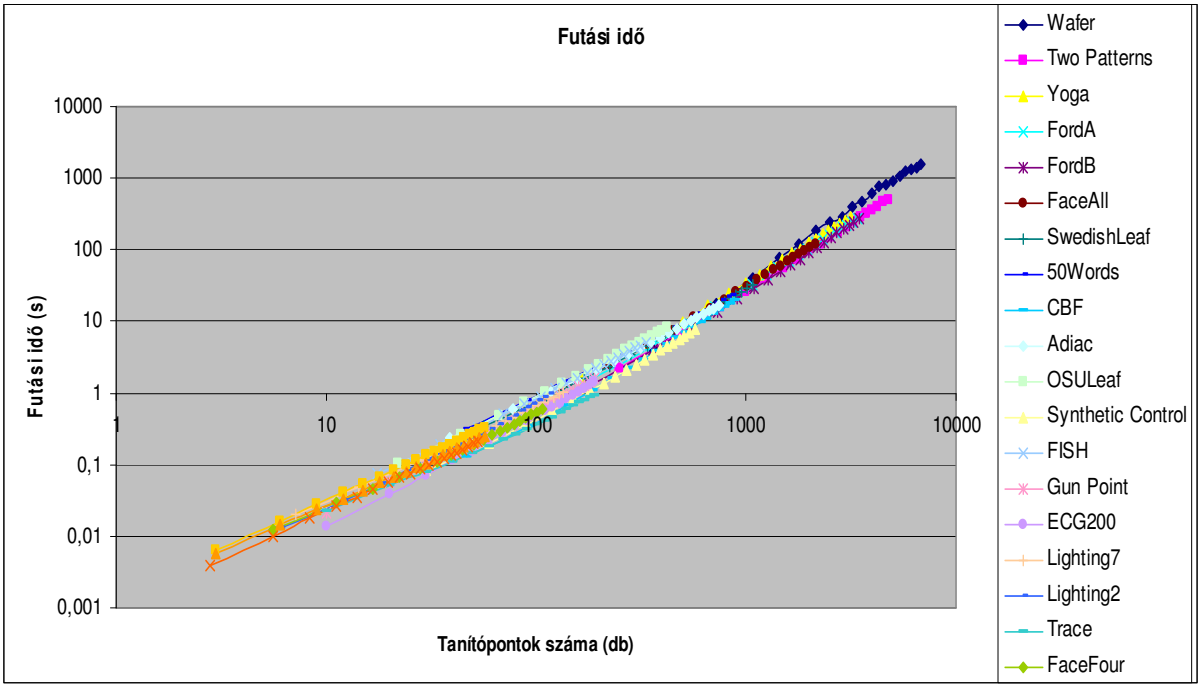
- Processzor: Core 2 Mobile T7200 @ 2.0GHz
- Memória: 1 GB DDR2 @ 533MHz
- Operációs rendszer: Windows XP (SP3)
- TICK-ek száma másodpercenként: 3579545

4.4.4. A futási idő tanítópontok számától való függésének vizsgálata

Excel segítségével adatsoronként ábrázoltam a modellépítés futási idejét a tanítópontok számának és az adatsor méretének arányának függvényében. A görbék (általában) elég szabályosak voltak ránézésre, ezért különböző polinom függvényeket illesztettem rájuk az Excel segítségével.

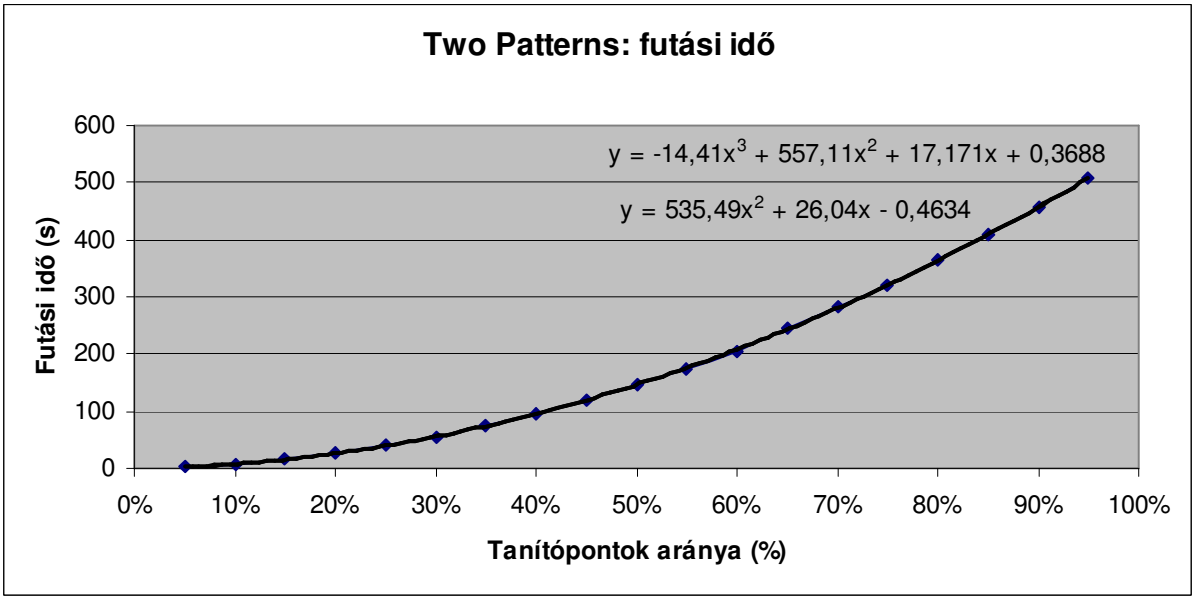
A 4.6. ábrán látható összes görbe, a Wafer görbéjének kivételével. (Ez utóbb a többiektől jelentősen eltérően néz ki, róla a 4.4.4.3. fejezetben lesz szó). A tengelyeken az ábrázolás azért logaritmikus, mert az egyes adatsorok idősorainak a száma és így a tanítási idők nagyságrendileg eltérnek.

Látható, hogy a görbék hasonlóak, csak az együtthatókban van eltérés. A logaritmikus skálán a görbék kb. egyenesek, azaz normál skálán valamilyen polinom függvényre hasonlítanak.



4.6. ábra: Az összes adatsor futási idő görbéje a Wafer kivételével

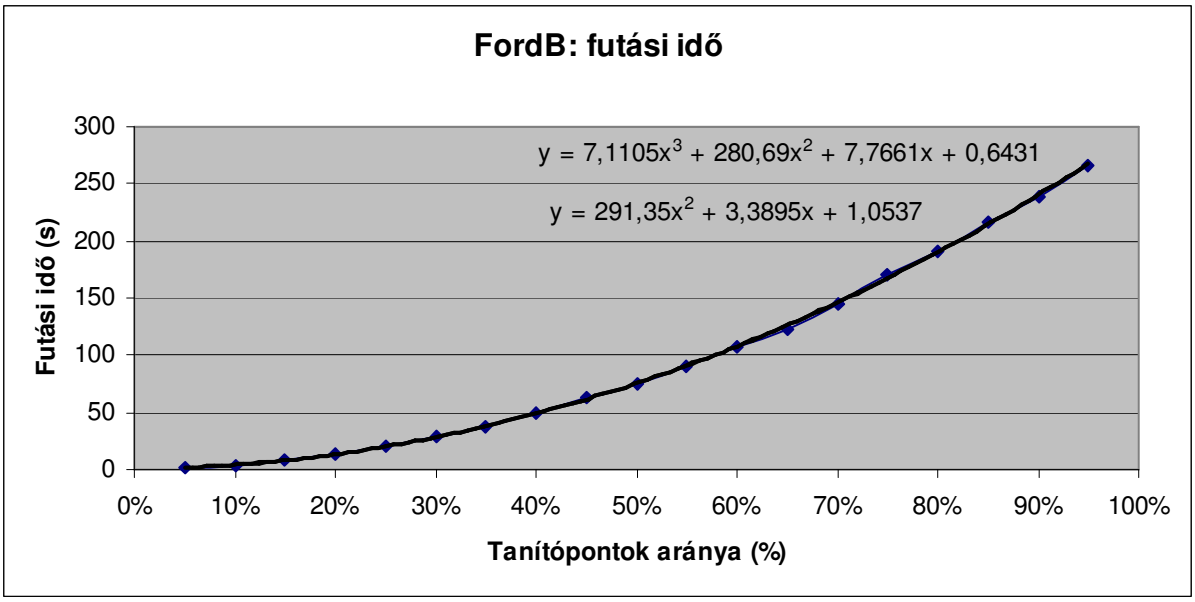
4.4.4.1. Másodfokú függvényre hasonlító görbék



4.7. ábra: A Two Patterns adatsor futási idő görbéje

A legtöbb görbe úgy néz ki, mint egy egyszerűbb polinom függvény. Ilyen a Two Patterns adatsor görbéje is, ami a 4.7. ábrán szerepel. Első ránézésre másodfokúnak tűnik. Az Excel által illesztett másodfokú görbe elég jól fedi az ábrázolt pontokat.

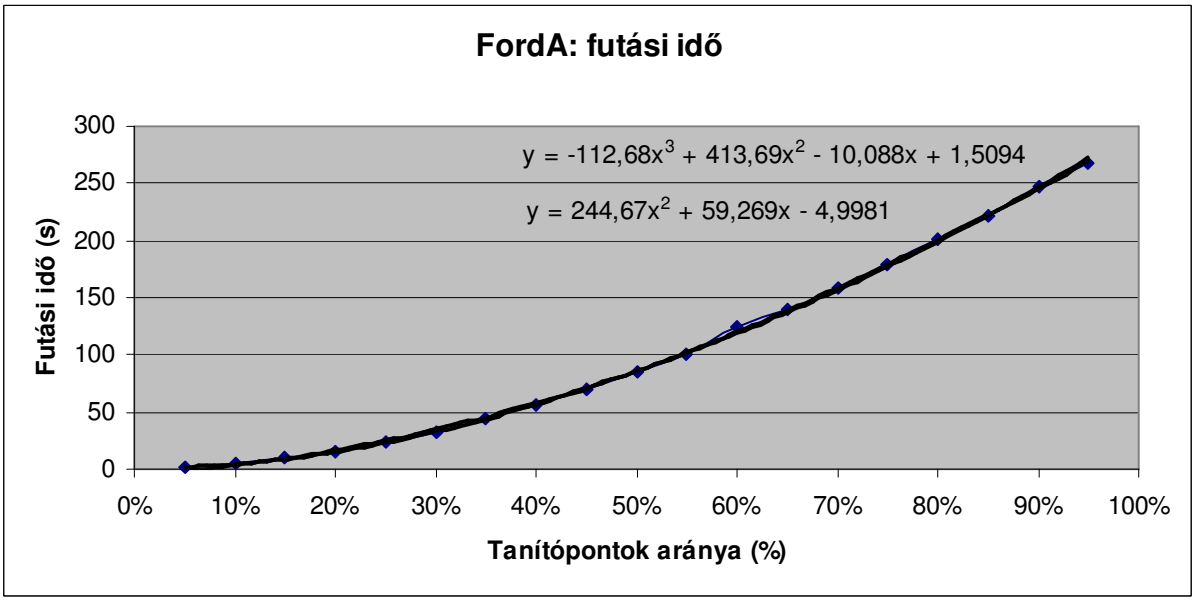
Harmadfokú görbe illesztésénél látszik, hogy a harmadfokú tag jelentéktelen a többihez képest, azaz a görbe valóban másodfokú. Ez lehetséges annak ellenére, hogy a csomópontonkénti időfüggvény az idősorok számának köbével arányos. Egyszerűen annyi történik, hogy a fa szerkezete és a mélysége együttesen kiejti valamennyire a harmadfokú tagot. A mérések azt igazolják, hogy ez az általános eset. Igazából nem tisztán másodfokú függvényről beszélünk, inkább azt mondjuk, hogy a függvény jobban hasonlít egy másodfokú függvényre, mint egy harmadfokúra, vagy egy negyedfokúra.



4.8. ábra: A FordB adatsor futási idő görbéje

A legtöbb görbe hasonlóan néz ki, persze a görbe egyenlete eltérhet a Two Patterns-étól. A FordB görbéje (4.8. ábra) szintén egyértelműen másodfokú, de kevésbé görbül, mint az előző.

4.4.4.2. Harmadfokú függvényre hasonlító görbék

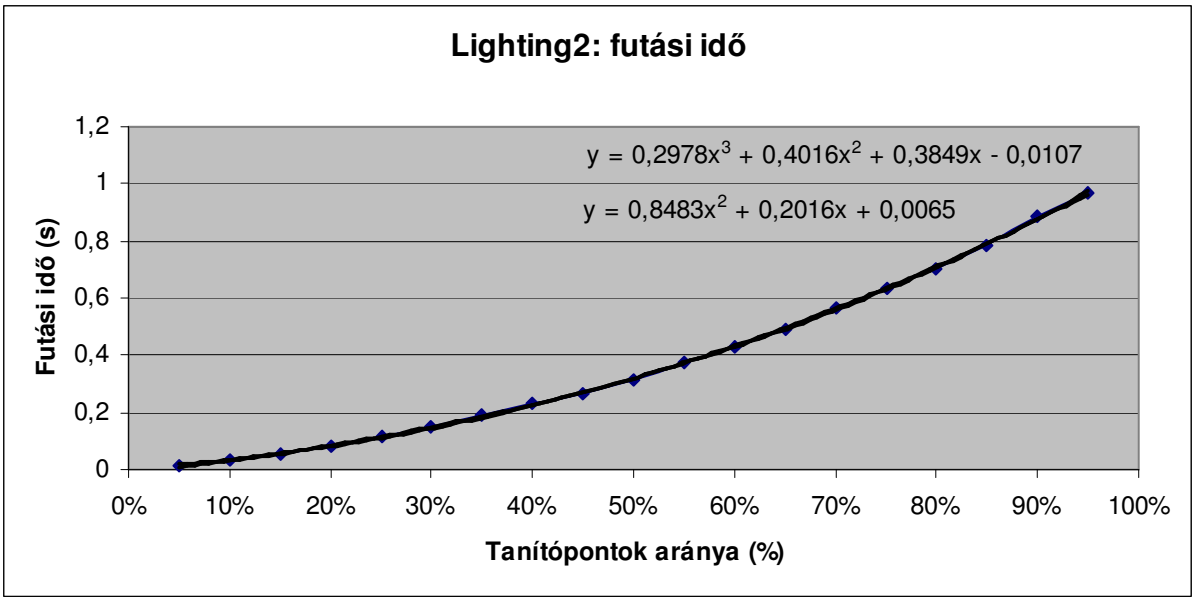


4.9. ábra: A FordA adatsor futási idő görbéje

Vannak olyan adatsorok, mint például a FordA (4.9. ábra) amiknek a görbéjéről nehezen dönthető el, hogy inkább másodfokú, vagy inkább harmadfokú. Mint látható, itt a harmadfokú tag együtthatója azonos nagyságrendű, mint a másodfokúé.

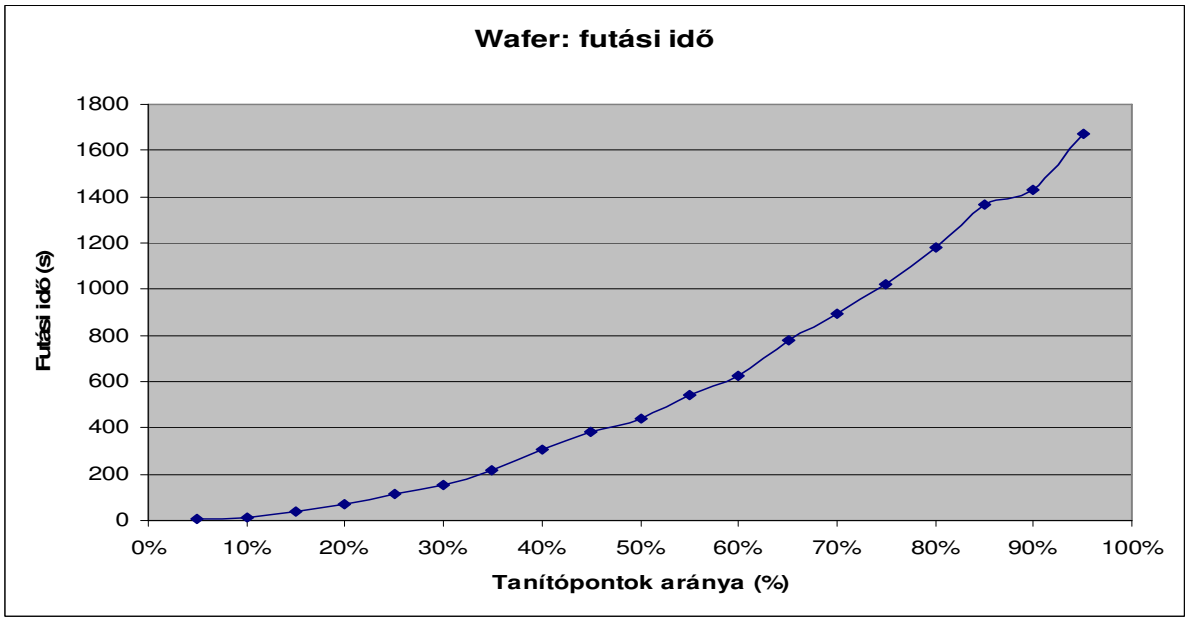
Ugyanez a helyzet a említett Lighting2 görbéjével (4.10. ábra), csak az együtthatók nagysága kisebb, mint a FordA esetében, hiszen gyorsabb a modellépítés, mert az összes pont, és így a tanítópontok száma kisebb mint a FordA esetében.

Az említett FordA és a Lighting2 adatsorok mellett a Lighting7 görbéjére jellemző, hogy a harmadfokú tag együtthatója azonos nagyságrendű, mint a másodfokúé.



4.10. ábra: A Lighting2 adatsor futási idő görbéje

4.4.4.2. A Wafer-anomália és magyarázata



4.11. ábra: A Wafer adatsor futási idő görbéje

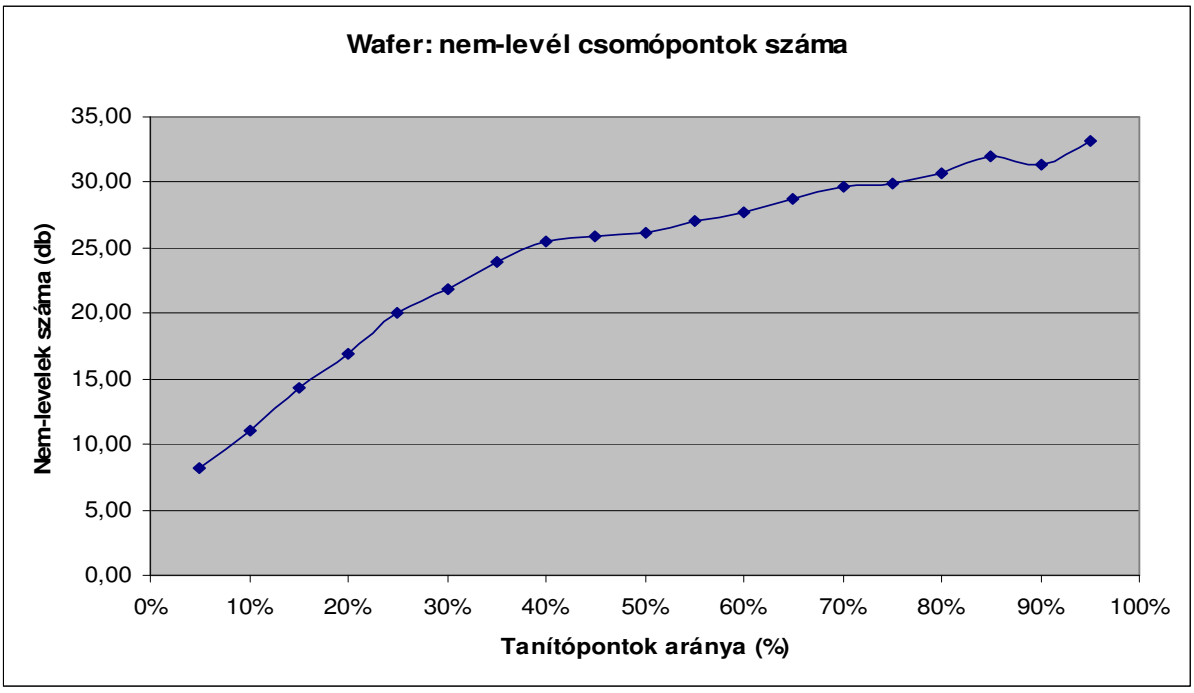
A Wafer görbéje eltér a többitől, ez látható a 4.11. ábrán. Mint látható, a görbe szabálytalan, főként a vége felé. A Wafer más miatt is speciális adatsor: itt a legtöbb az idősorok száma (7164), ezzel magasan megelőzi az 5000-es méretű Two Patterns adatsort, ami a második legnagyobb. A több helyen megfigyelhető törések miatt még egy mérést végeztem az adatsorral az esetleges mérési hiba kiküszöbölésére, de az eredmény ugyanez lett.

A töréseket az adatsor nagy mérete okozza: a ShiftTree nagy mennyiségű többlet információhoz jut hozzá két tanítási szint között. Így könnyebben megtalál egy optimálisabb feltételt, ami homogén csomópontokat eredményez, az előző tanításoknál magasabb szinten, így teljes ágak végigszámolása alól mentesül. Azaz hiába több pontra kell számolni, a fa kisebb lesz, ezért a futási idő növekedése jelentősen kisebb, mint amennyi változatlan szerkezetnél lenne. Ezt látjuk törések formájában. A görbén több ilyen törés is megfigyelhető, a legjelentősebb 85% és 90% között van.

Szintek:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
5%	1	2	3,8	4,9	4,4	1,4	-	-	-	-	-	-	-	-
10%	1	2	4	5,7	6,5	3,2	0,8	-	-	-	-	-	-	-
15%	1	2	4	6,1	8,5	5,3	2	0,8	-	-	-	-	-	-
20%	1	2	4	6,1	9,2	7	3,3	1,5	0,6	0,2	-	-	-	-
25%	1	2	4	6,6	9,9	9,1	4,8	2,2	1,4	-	-	-	-	-
30%	1	2	4	6,9	10,8	10,8	4,5	2,3	1,4	0,5	0,4	-	-	-
35%	1	2	4	7	10,7	11,4	6,8	3,8	1,5	0,4	0,2	0,2	-	-
40%	1	2	4	7	10,9	12,2	7,3	4,4	2,5	0,4	0,2	-	-	-
45%	1	2	4	7	11,1	12,2	7,6	4,9	2,3	0,8	-	-	-	-
50%	1	2	4	7,1	11,3	12,1	7,7	3,9	2,2	1,2	0,7	0,2	-	-
55%	1	2	4	7,1	10,9	12,6	8,4	4,4	2,3	0,9	0,8	0,6	0,1	-
60%	1	2	4	7,2	11,3	13,1	9	4,4	2,5	1,1	0,5	0,1	0,1	0,1
65%	1	2	4	7,1	11,3	13,3	9,4	4,5	3	2,1	0,7	0,1	-	-
70%	1	2	4	7,3	11,5	13,4	9,6	5,7	3,2	1,9	0,7	-	-	-
75%	1	2	4	7,5	11,7	13,4	9,9	5,3	2,7	1,3	1	0,5	0,4	0,2
80%	1	2	4	7,6	11,8	13,6	10	5,4	3,7	1,9	0,9	0,4	0,2	-
85%	1	2	4	7,7	11,9	13,5	10,6	5,8	4,2	2,6	0,9	0,6	0,1	-
90%	1	2	4	7,8	12	14	10,2	5,7	4,9	2,1	0,1	-	-	-
95%	1	2	4	7,9	11,9	14,8	9,6	6,7	7,1	2,3	-	-	-	-

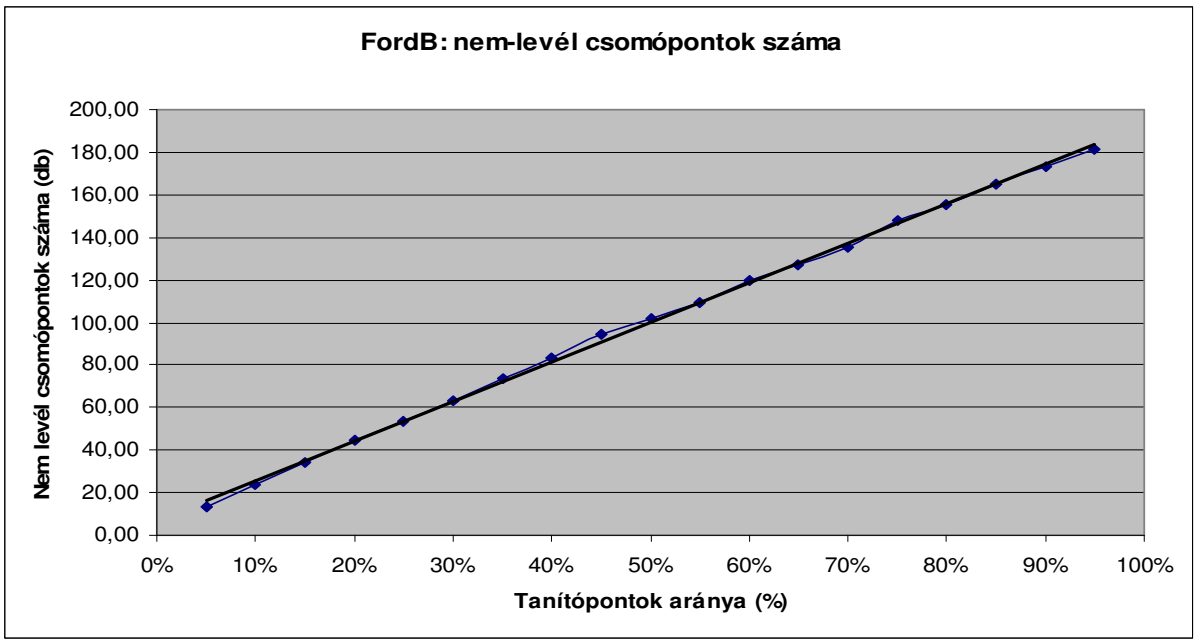
4.11. táblázat: A Wafer modellek átlagos csomópontszámai

A feltevés igazolására megnéztem, hogy az egyes százalékos szinteken átlagosan hogyan néz ki a fa felépítése, azaz szintenként hány csomópontja van a fának (4.11. táblázat). Látható, hogy az említett 85% és 90% között visszaesés van a csomópontok számában.



4.12. ábra: A Wafer adatsorhoz épített modellek átlagos belső csomópontszám görbéje

Mivel az algoritmus csak a nem-levél csomópontokban számol, számunkra is ezek az érdekesek, ezért ábrázoltam a nem-levél csomópontok számát a tanítópontok arányának függvényében a 4.12. ábrán. Itt is megfigyelhető a visszaesés 85% és 90% között. Több visszaesést nem találunk, ugyanakkor az is látható, hogy ez a görbe is igen szabálytalan. Azok a szakaszok, ahol a görbe alig emelkedik, olyan váltások, ahol a mérések egy részében visszaesés volt, a többiben pedig növekedés, és ezek kiejtették egymást az átlagolás során.



4.13. ábra: A FordB adatsorhoz épített modellek átlagos belső csomópontszám görbéje

Összehasonlítva a nem-levél görbét a FordB adatsor ugyanilyen görbéjével (4.13. ábra), látszik, hogy az anomália tényleg összefüggésben van a csomópontok számának növekedésével. Általános esetben a nem-levél csomópontok átlagos száma lineárisan nő a hozzáadott tanítópontokkal, vagy egy adott ponton túl stagnál, mint a Trace adatsor esetén.

Tanítópontok	Összes csomópont	Nem-levél csomópontok	Futási idő
5%	21	10	4,91018
10%	29	14	16,7036
15%	31	15	34,6528
20%	41	20	74,0912
25%	**39**	**19**	133,482
30%	49	24	176,064
35%	59	29	249,946
40%	63	31	503,37
45%	**55**	**27**	**366,664**
50%	**53**	**26**	**353,689**
55%	57	28	467,186
60%	57	28	618,466
65%	77	38	1029,86
70%	**61**	**30**	**843,85**
75%	**53**	**26**	**839,795**
80%	59	29	960,002
85%	59	29	1081,21
90%	59	29	1205,859012
95%	**53**	**26**	1345,958032

4.12. táblázat: Egy Wafer modell összes és belső csomópontjainak a száma és futási idői

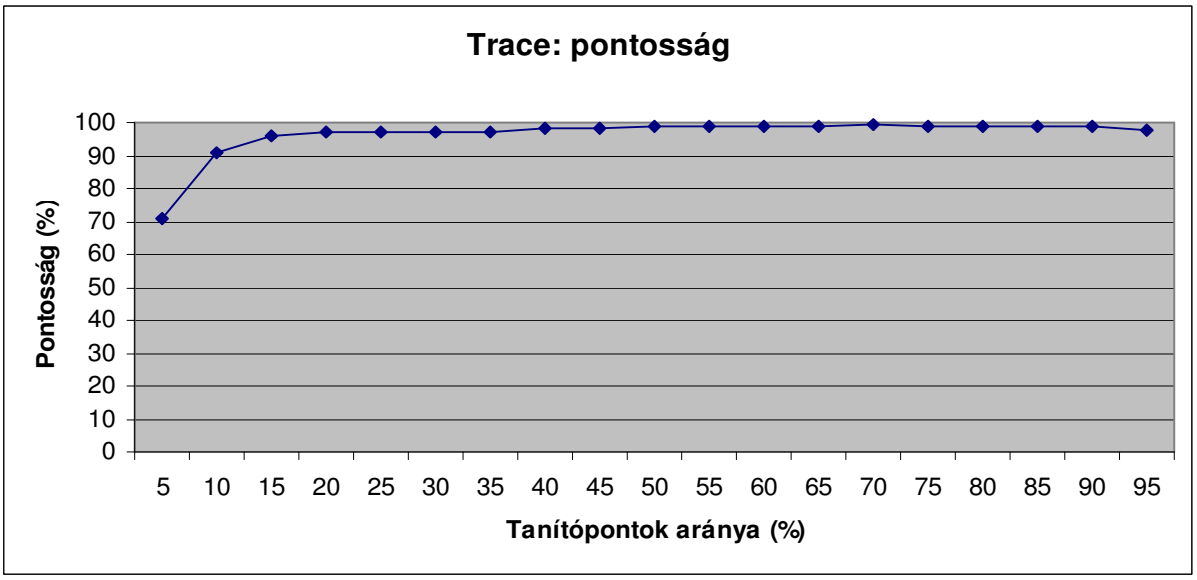
Egy konkrét modellt vizsgálva minden szinten (4.12. táblázat) látható, hogy szinte mindenhol, ahol visszaesés van a nem-levél csomópontok számában, ott visszaesés tapasztalható a futási időben. A kiátlagolt ábrán is ezek jelennek meg törések formájában: az átlagot csökkenti egy-egy visszaesett futási idő, ami által a görbe növekedése kisebb lesz, mint az elvárt.

A jelenség más nagy adatsoroknál is jelentkezhet. A FordA görbéjén (4.9. ábra) is megfigyelhető egy kisebb törés 20% és 30% között.

4.4.5. A pontosságának a tanítópontok számától való függésének vizsgálata

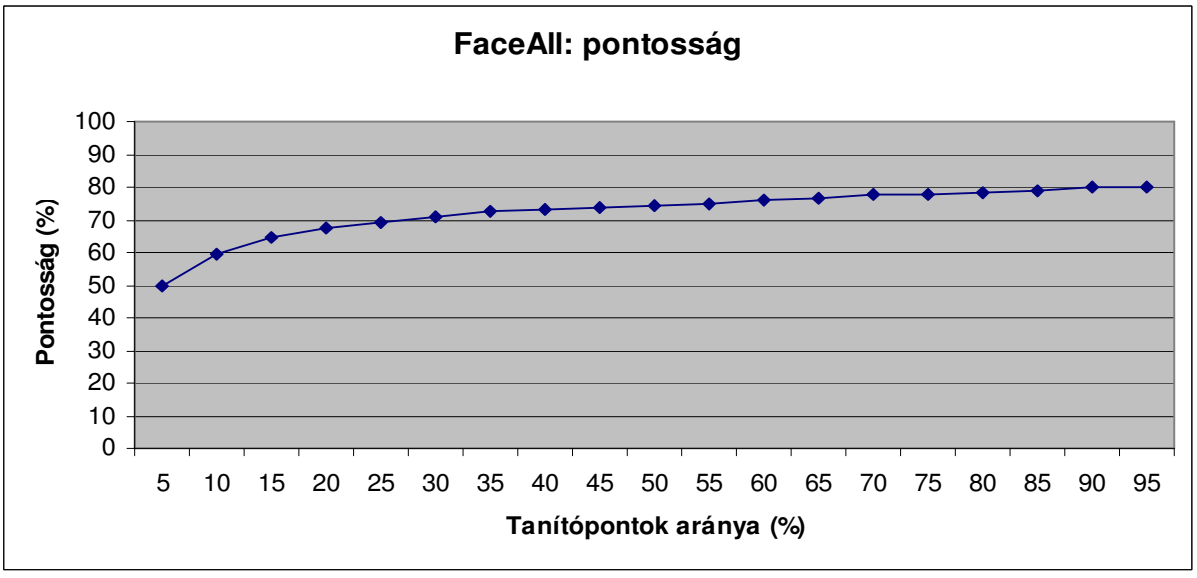
Míg a futási idők görbéi nagyon hasonlóak voltak, a Wafer kivételével az összes adatsornál, a pontosság görbék jelentősen eltérnek egymástól. Ennek az az oka, hogy az adatsorok mérete és egyéb tulajdonságaik jelentősen eltérnek. Ez a pontosságnál jelentős hatással van a görbék alakjára is. A görbéket négy csoportba soroltam az alakjuk alapján, ezeket ismertetem a következő négy pontban, majd az alfejezet végén az összes adatsort besorolom a kategóriákba, és jellemzem azokat.

4.4.5.1. Normál görbék



4.14. ábra: A Trace adatsor pontosság görbéje

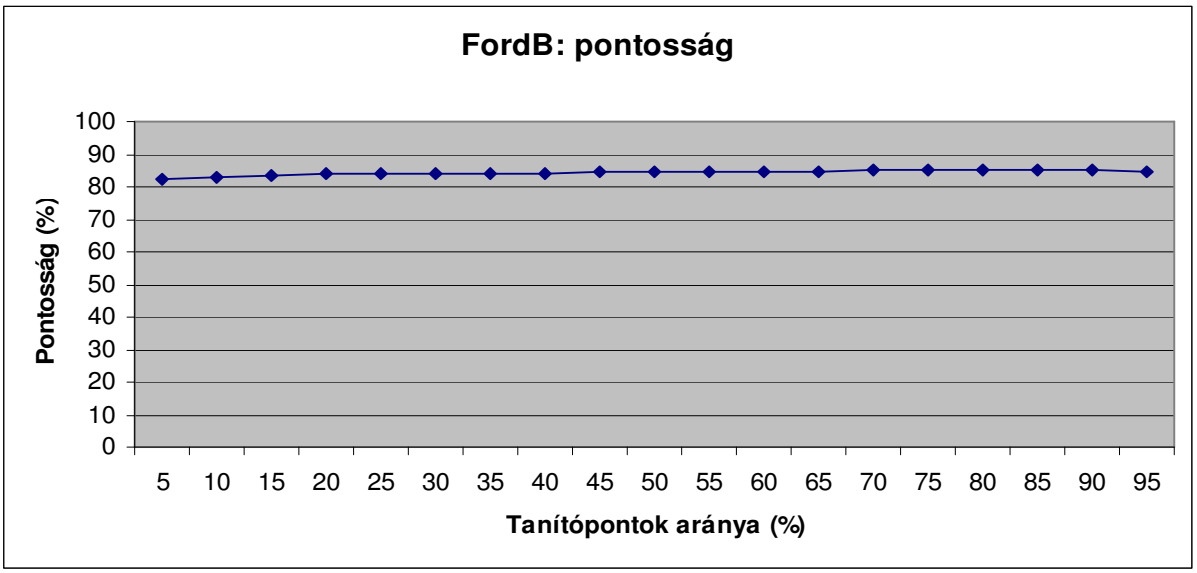
A Trace pontosság görbéje (4.14. ábra) egy, az elvárásoknak megfelelően „viselkedő” pontosság görbe: kis mennyiségű tanítópont mellett gyengébben teljesít, majd egy adott mennyiség után beáll a végső pontosságának környezetébe. A függvény alakja leginkább logaritmus, vagy gyökfüggvényre emlékeztet, ami logikus is, hiszen az 5%-ról a 10%-ra való ugrás sokkal fontosabb a modellépítés helyességét szem előtt tartva, mint a 75%-ról a 80%-ra ugrás. A normál görbék csoportjába tartozó függvényeket ez az alak jellemzi.



4.15. ábra: A FaceAll adatsor pontosság görbéje

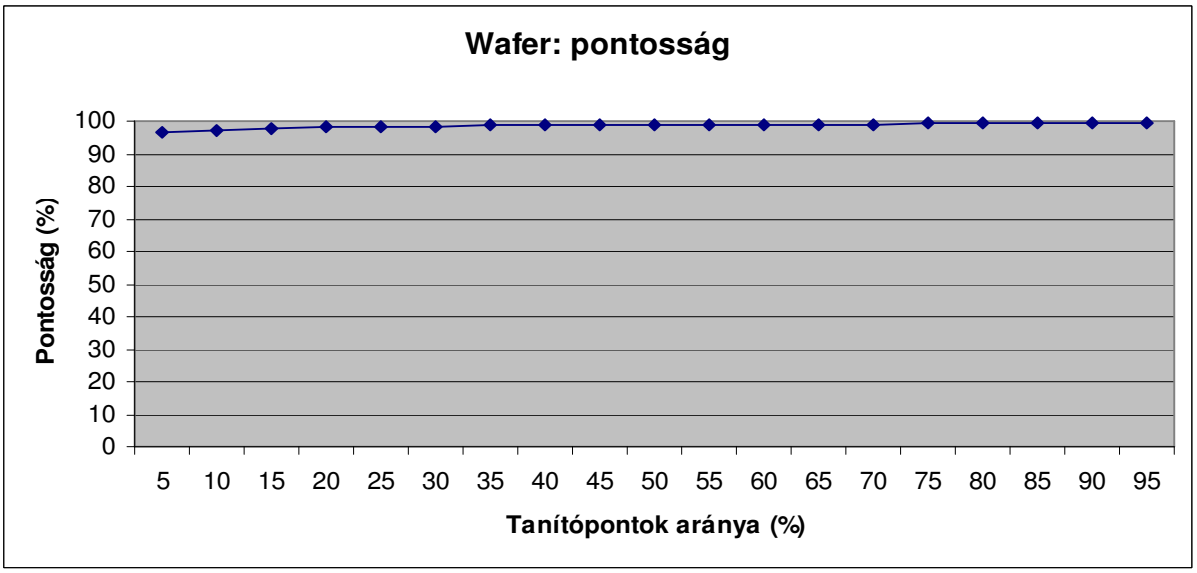
A FaceAll görbéje (4.15. ábra) is egy normál görbe, de láthatjuk rajta, hogy jóval lassabban áll be a pontosság határhoz, azaz az adatsor nehezebben tanulható. Az is megfigyelhető, hogy a pontosság felső határa itt nem 100%, hanem 80% körül van. Ez a felső határ adatsoronként és konfigurációként eltérő nagyságú lehet, és azt jelzi, hogy egy adott konfiguráció mellett a ShiftTree mennyire tudta megtanulni a teljes adatsor belső összefüggéseit.

4.4.5.2. Egyenletes görbék



4.16. ábra: A FordB adatsor pontosság görbéje

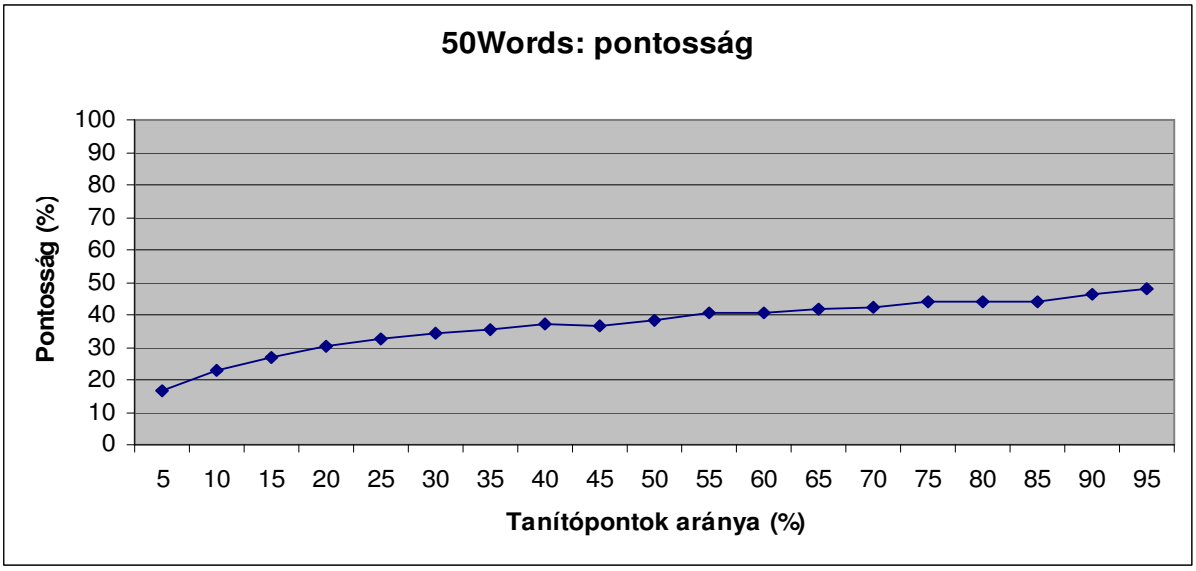
A FordB görbéje (4.16. ábra) szinte egy vízszintes vonal valahol 82% és 86% között. Azaz már az idősorok kis részén tanított modellek pontosságai is a pontosság felső határának környezetében vannak. Az ilyen egyenletes teljesítménynek az lehet az oka, hogy az adatsor sok idősort tartalmaz, – a FordB esetében 3636 idősort – és csak kevés – a FordB-nél 2 – osztályba kell sorolni az adatokat. Ezért már 5%-10% tanítópontnál elég sok tanítópont áll rendelkezésünkre mindkét osztályból az összefüggések pontos megtanulásához.



4.17. ábra: A Wafer adatsor pontosság görbéje

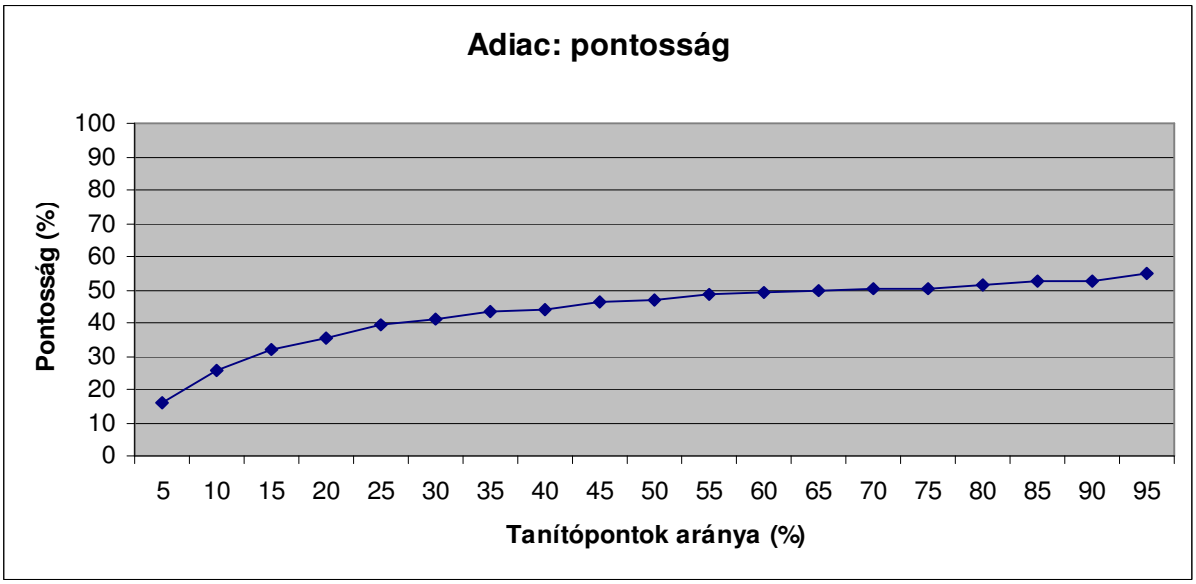
Nem meglepő, hogy a legnagyobb és szintén két osztállyal rendelkező adatsor, a Wafer, pontossággörbéje (4.17. ábra) is egyenletes, csak még kisebb a vonal íve, mivel több a tanítópontok száma.

4.4.5.3. Növekvő görbék



4.18. ábra: Az 50Word adatsor pontosság görbéje

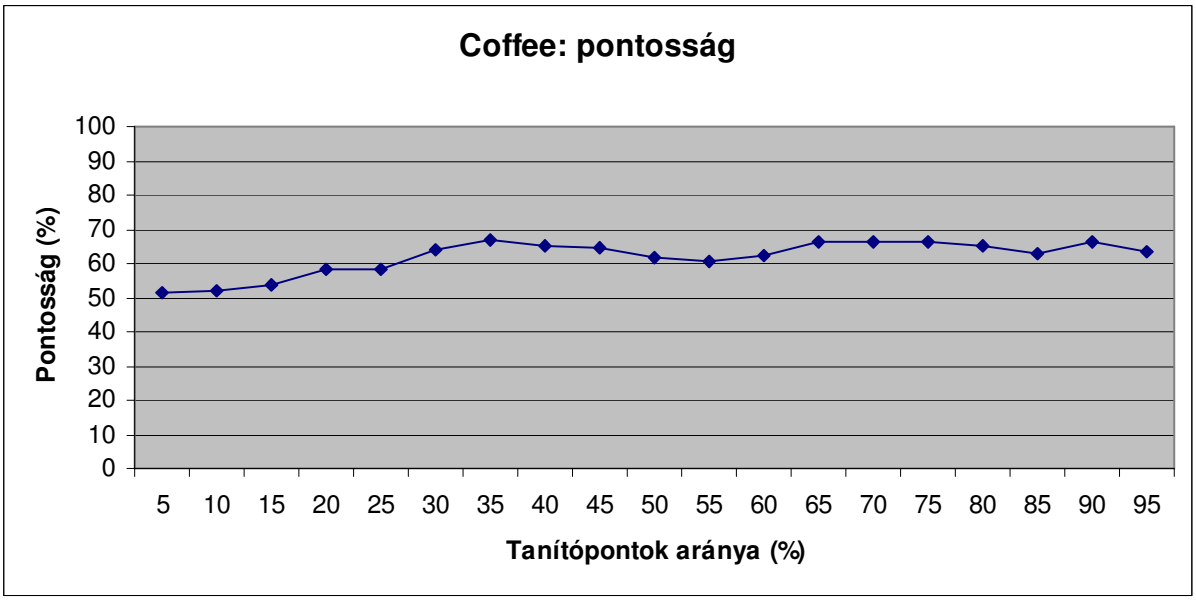
Az 50Words görbéje (4.18. ábra) az eddigiektől jelentősen eltérő, mivel a vége felé megfigyelhető egy jelentősebb emelkedés, mintha a görbe folytatása a tanítópontok 100%-án túl még növekedne. Ennek az lehet az oka, hogy bár összesen 905 idősor található az 50Words adatsorban, az 50 osztályhoz ez ‚egy ilyen tanulás-intenzív módszernek, mint a ShiftTree, még kevés, mert még a 95%-os tanításnál is csak átlagosan 17 tanítópontot jelent osztályonként. Ha az adatsor elegendően több idősort tartalmazna, azaz a tanítópont-hiány megszűnne, akkor ez a görbe is a normál görbék csoportjába tartozna, ez az összes növekvő görbe közös jellemzője.



4.19. ábra: Az Adiac adatsor pontosság görbéje

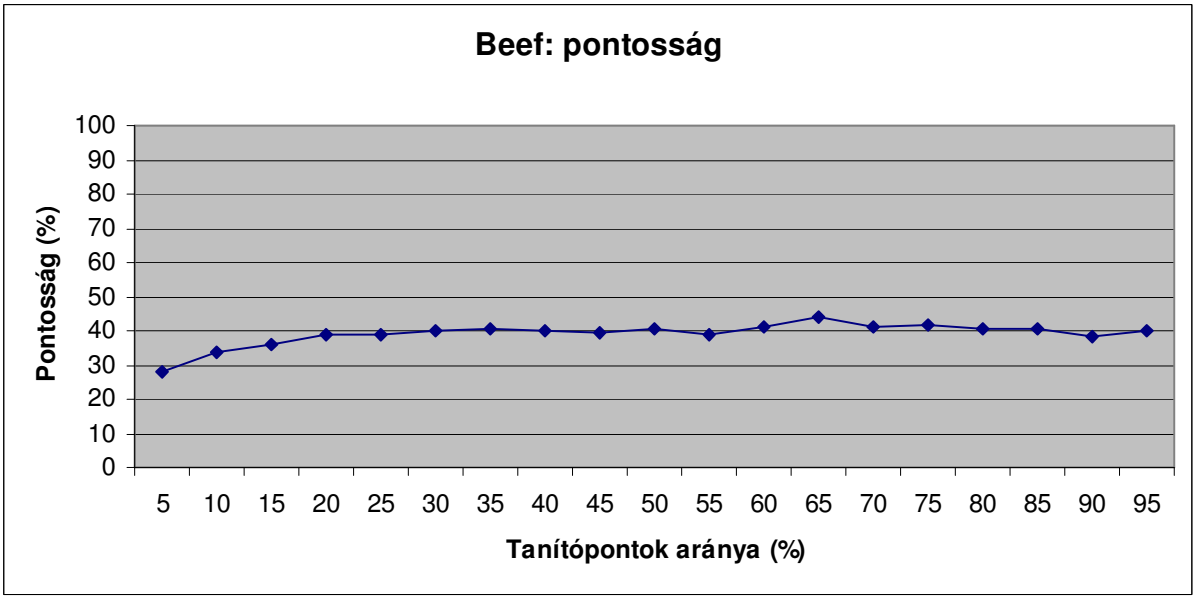
Hasonlóan néz ki az Adiac adatsor görbéje (4.19. ábra), mivel az adatsor tulajdonságai is hasonlóak: 781 idősor, 37 osztály, 20 tanítópont átlagosan osztályonként 95%-os tanításnál. Az is látszik, hogy valamennyivel jobb a helyzet, mint az 50Words esetében, azaz a görbe kevésbé látványosan növekszik a vége felé, mivel több mintából tanulhat osztályonként az algoritmus.

4.4.5.4. Hullámzó görbék



4.20. ábra: A Coffee adatsor pontosság görbéje

A Coffee pontosság görbéjében (4.20. ábra) jelentős hullámzás figyelhető meg. Ennek az az oka, hogy az egész adatsor csak 56 idősort tartalmaz. Ebben az esetben túl kevés a teszteléshez felhasznált pontok száma, ezért egy-egy félresorolt idősor jelentősen rontja a pontosságot. 60%-os tanításnál átlagosan 22,4; 95%-os tanításnál 2,8 ponton tesztelünk, ami jelentősen növeli a félresorolt pontok hatását. Ezt az is alátámasztja, hogy amíg a Trace adatsor esetében 60%-os tanítás esetén a pontosságértékek szórása 0,82 az Adiac esetében 3,48 addig a Coffee esetében 7,17. 95%-os tanítás esetében még szembetűnőbb a különbség: Coffee – 30,4; Adiac – 8; Trace – 4.



4.21. ábra: A Beef adatsor pontosság görbéje

Kisebb hullámzás megfigyelhető a szintén kicsi – 60 idősort tartalmazó – Beef adatsor görbéjén (4.21. ábra) is. A Beef adatsor pontosságának a szórása szintén elég magas, 95%-os tanításnál 25,59.

4.4.5.5. Az adatsorok csoportosítása a görbéjük alapján

Fentebb látható volt a négy kategória leírása. Ezek a kategóriák természetesen átfedésben lehetnek egymással, mert egy görbéről nem tudjuk mindig biztosan eldönteni, hogy melyik kategóriába tartozik. Fontos, hogy alapvető jellemzőik miatt az egyenletes kategória nincs átfedésben a növekvő és a hullámzó görbék kategóriájával. Ebben a pontban mind a 22 adatsort besorolom egy vagy több kategóriába, és megvizsgálom, hogy miért kerültek oda.

4.4.5.5.1. Az adatsorok néhány tulajdonsága:

A kategóriák vizsgálatához az adatsorok néhány alapvető tulajdonságát kiemeltem. Ezek a 4.13. táblázatban láthatóak. Az egyes adatsorokhoz tartozó oszlopokat külön színnel jelöltem, amik megfelelnek a 4.22. ábrán a kategóriákat, illetve azok metszetét jelző színekkel, aszerint, hogy melyik adatsor hová tartozik.

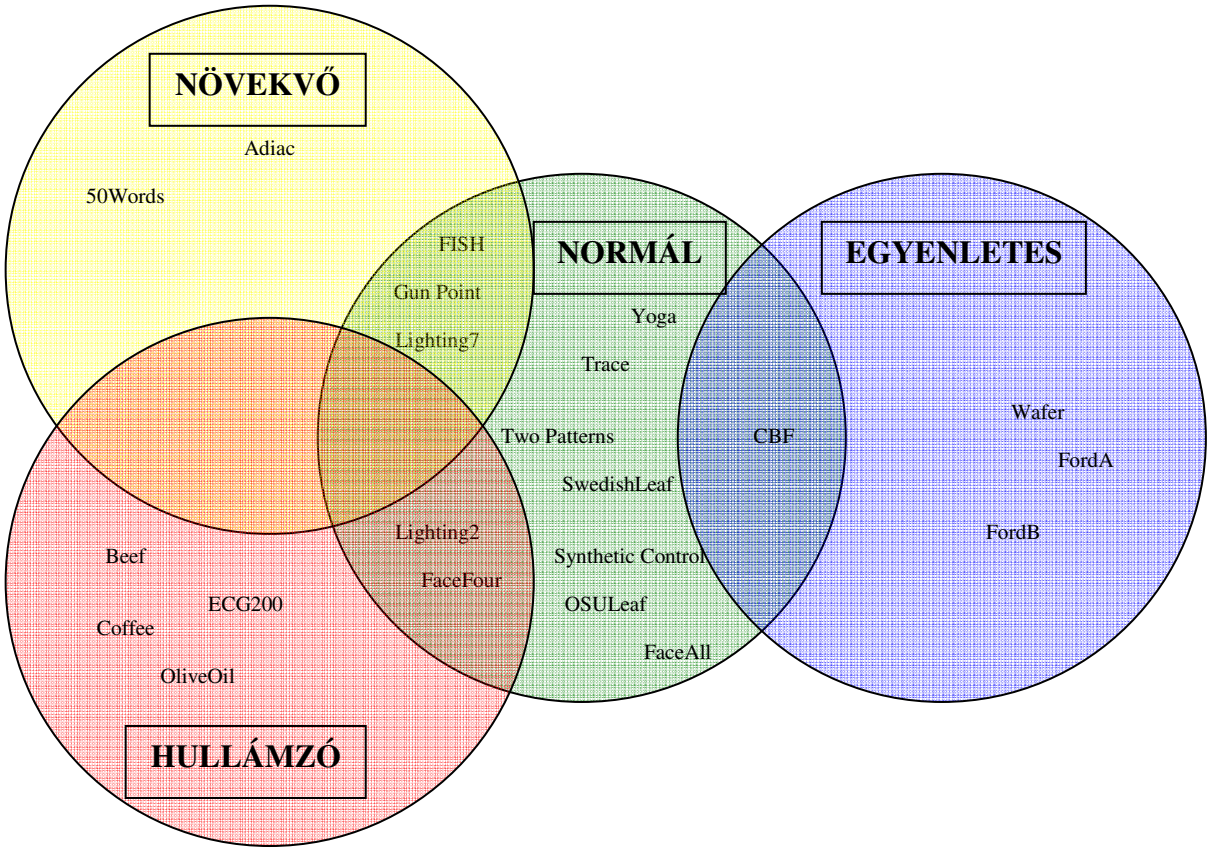
A színek jelentése: Halványsárga – Növekvő;
Halványpiros – Hullámzó;
Halványzöld – Normál;
Halványlila – Egyenletes;
Narancssárga – Hullámzó és Normál metszete;
Sárgászöld – Növekvő és Normál metszete;
Élénk kék – Egyenletes és Normál metszete.

	50Words	**Adiac**	**Beef**	**CBF**	**Coffee**	**ECG200**
Osztályok száma	50	37	5	3	2	2
Adatsor mérete	905	781	60	930	56	200
Méret/osztályok	18,1	20,11	12	310	28	100
Idősor hossza	270	176	470	128	286	96
Szórás 50%-on	2,32	2,58	9,42	1,29	7,05	3,15
Szórás 95%-on	6,3	8,03	25,59	3,13	30,4	13,76
	FaceAll	**FaceFour**	**FISH**	**FordA**	**FordB**	**GunPoint**
Osztályok száma	14	4	7	2	2	2
Adatsor mérete	2250	112	350	3571	3636	200
Méret/osztályok	160,71	28	50	1785,5	1818	100
Idősor hossza	131	350	463	500	500	150
Szórás 50%-on	1,45	3,75	4,51	1,19	0,79	2,86
Szórás 95%-on	3,78	18,68	12,38	2,99	2,55	7,68
	Lighting2	**Lighting7**	**OliveOil**	**OSULeaf**	**SwedishLeaf**	
Osztályok száma	2	7	4	6	15	
Adatsor mérete	121	143	60	442	1125	
Méret/osztályok	60,5	20,43	15	73,67	75	
Idősor hossza	637	319	570	427	128	
Szórás 50%-on	5,85	6,98	8,59	3,71	2,45	
Szórás 95%-on	15,24	18,83	32,26	7,83	6,59	
	SyntheticControl	**Trace**	**TwoPatterns**	**Wafer**	**Yoga**	
Osztályok száma	6	4	4	2	2	
Adatsor mérete	600	200	5000	7174	3300	
Méret/osztályok	100	50	1250	3587	1650	
Idősor hossza	60	275	128	152	426	
Szórás 50%-on	1,78	1,27	0,89	0,26	1,55	
Szórás 95%-on	3,82	4,1	0,93	0,43	3,36	

4.13. táblázat: Az adatsorok néhány tulajdonsága

4.4.5.5.2. Kategóriák:

A 4.22. ábra ábrázolja az egyes kategóriákat, és a kategóriákba tartozó adatsorokat.



4.22. ábra: Az adatsorok elhelyezkedése a kategóriákban

Az egyes kategóriákat az alábbiak jellemzik:

- A növekvő kategóriában olyan adatsorok vannak, amelyeknek bár összesen sok tanítópontjuk van, az osztályonkénti tanítópontok száma kevés, ezért a pontosságon még lehetne javítani új tanítómintákkal.
- A hullámzó kategóriában főként olyan adatsorok vannak, ahol kevés az összes idősor száma, ezért a teszteléshez használt pontok száma is nagyon alacsony, ami miatt pontosságok szórása jelentős, főként a magasabb százalékos tanításoknál.
- Az egyenletes kategóriába azok az adatsorok kerültek, amelyek kellően nagy számú idősorral, és kevés osztállyal rendelkeznek, ezért kis százalékos tanításnál is a végső pontossághatár környezetébe eső pontosságú modellt eredményeznek.
- A normál kategóriába került az összes olyan adatsor, aminek nincs elég tanítópontja ahhoz, hogy egyenletes pontosság görbéje legyen, de annál több idősora és osztályonkénti pontja van, hogy a hullámzó és/vagy a növekvő kategóriákba tartozzon.

4.4.5.5.3. Kivételek

A látható tulajdonságok (méret, osztályszám, stb.) mellett nehezen megfigyelhető tulajdonságok is befolyásolják azt, hogy milyen lesz a pontosság görbe alakja. Például az ECG200 adatsor összes pontja nem annyira kevés, mégis elég nagy szórása van magasabb százalékos tanításoknál. A CBF annyira nem nagy adatsor, mint a Yoga, mégis a CBF görbéje tekinthető egyenletesnek is, míg a Yoga görbéje tisztán a normál kategóriába tartozik. Ezeket a szabályoktól való eltéréseket az adatsorok tanulhatósága okozhatja. Tehát az, hogy a ShiftTree az adott konfigurációval hogyan képes megfogni egy-egy adatsor lényegét, jelentősen eltolhatja a kategóriák határfeltételeit.

5. Többattribútumos idősorok osztályozásának tesztelése

A 2. fejezetben láthattuk, hogy az algoritmus alkalmas a többváltozós idősorok osztályozására is a 2.4. pontban említett kiegészítések alkalmazásával. Ebben a fejezetben a módszert többváltozós idősorokon tesztelem, érintve minden fontosabb kritériumot: pontosság mértéke fix tanító- és teszthalmaz mellett, keresztvalidációs mérés a futási időre és a pontosságra, ezek összehasonlítása az egyváltozós esettel, stb.

A fejezetet a teszteléshez használt adatok ismertetésével kezdem, majd mindkét adatsor eredményeinek egy-egy külön alfejezetet szentelek.

5.1. A teszteléshez használt adatok

A teszteléshez két adatsort használtam. Az egyik ezek közül digitalizált hangokat tartalmaz, a másik pedig egy mobiltelefonba épített gyorsulásmérő adatait.

5.1.1. Az AE adatsor leírása

Az AE adatsor digitalizált hangokról tartalmaz információkat. Az adatsor úgy áll elő, hogy 9 különböző férfi a japán ae hangot mondta ki [21]. Egy ilyen kiejtéshez 12 változó tartozik, amelyek egy többváltozós idősort alkotnak. Az egyes idősorok hossza nem feltétlenül ugyanakkora, ezért jól teszteli a módszer rugalmasságát is. A feladat annak az eldöntése az adatok alapján, hogy éppen ki „beszél”.

Az adatsor tanító- és teszthalmazának tulajdonságai az 5.1. táblázatban láthatóak.

	Tanítóhalmaz	Teszthalmaz
Idősorok hossza	7-29	
Osztályok száma	9	
Változók száma	12	
Idősorok száma	270	370
Osztályok pontjainak eloszlása	30,30,30,30,30,30,30,30,30	31,35,88,44,29,24,40,50,29

5.1. táblázat: Az AE adatsor tulajdonságai

5.1.2. A gyorsulásmérő adatsor

A gyorsulásmérő adatsor egy telefonba épített gyorsulásmérő adatait tartalmazza, amivel 10 különböző gesztust írt le 4 különböző ember [22]. Ezen kívül van néhány zajszerű idősor is. Minden idősornak 3 változója van, amik megfelelnek a gyorsulásmérő által mért 3 gyorsulásértéknek.

Az eredeti osztályozási feladat az, hogy ismerjük fel az egyes gesztusokat. Emellett az is értelmes osztályozási feladat, hogy találjuk ki, hogy éppen ki használja a telefont.

Az adatsor nem volt szétbontva tanító- és teszthalmazra. A teljes adatsor tulajdonságai az 5.2. táblázatban láthatóak.

Idősorok hossza	20-62 (zaj: 21-96)
Változók száma	3
Idősorok száma	502 +50 zaj
Osztályok száma gesztusfelismerésnél	10 +1 (zaj)
Osztályok pontjainak eloszlása gesztusfelismerésnél	50,50,50,50,51,50,50,50,50,51 +50 zaj
Osztályok száma személyfelismerésnél	4
Osztályok pontjainak eloszlása személyfelismerésnél	100,102,100,200

5.2. táblázat: A gyorsulásmérő adatsor tulajdonságai

5.2. Digitalizált hang adatok felismerése

Többféle tesztet végeztem az AE adatsorral. Végeztem egyszerű teszteket, ahol a két részre szedett (tanító és teszt) adatsort külön kezeltem, valamint végeztem a 4.4.1.1. pontban leírt keresztvalidációt is.

5.2.1. Egyszerű tesztek

Kétféle egyszerű tesztet csináltam. Az első esetben a tanítóhalmazon tanítottam és a teszthalmazon teszteltem, ez a normál teszt. A másik esetben a teszthalmazon tanítottam és a tanítóhalmazon teszteltem, ez a fordított teszt.

5.2.1.1 A program konfigurációja

Az egyszerű tesztekhez 3 különböző konfigurációt használtam. Az egyes konfigurációk között az eltérés a definiált CBO-k között, illetve azok sorrendjében volt. A program konfigurációi alább láthatóak.

Decider: DEntropy
ESO-k (sorrendben): NextMax, NextMin, Next(1), Next(2), Next(4), Next(8), Next(12), Next(16), Next(24), Next(32), Max, Min, Prev(1), Prev(2), Prev(4), Prev(8), Prev(12), Prev(16), Prev(24), Prev(32), PrevMax, PrevMin
1. CBO konfiguráció: Simple, Normal(0, 1, 0.05), Exp(1, 0.05), Normal(0, 0.5, 0.01), Exp(0.5, 0.01), Normal(0.5, 4, 0.01), Exp(2, 0.01)
2. CBO konfiguráció: Normal(0.5, 4, 0.01), Normal(0, 0.5, 0.01), Exp(2, 0.01), Exp(0.5, 0.01), Normal(0, 1, 0.05), Exp(1, 0.05), Simple
3. CBO konfiguráció: Simple, Normal(0, 1, 0.05), Exp(1, 0.05), Normal(0, 0.5, 0.01), Exp(0.5, 0.01), Normal(0.5, 4, 0.01), Exp(2, 0.01), CBEAverage(Normal(0.5, 4, 0.01)), CBEVariance(Normal(0.5, 4, 0.01))
4. CBO konfiguráció: Normal(0.5, 4, 0.01), Normal(0, 0.5, 0.01), Exp(2, 0.01), Exp(0.5, 0.01), Normal(0, 1, 0.05), Exp(1, 0.05), Simple, CBEAverage(Normal(0.5, 4, 0.01)), CBEVariance(Normal(0.5, 4, 0.01))
Információnyereség minimális határa: nincs
Szintkorlát: nincs

5.2.1.2. Normál teszt

Az 1. CBO konfiguráció mellett, a pontosság 82,97% lett. Összehasonlításképpen: az erre az adatsorra optimalizált algoritmus 94,1%-ot ért el [23], amihez közeli eredményt valószínűleg a ShiftTree optimalizálásával, esetleg új operátorok bevezetésével el lehetne érni.

A 2. CBO konfigurációval, ami előnyben részesíti a széles intervallumon történő súlyozott átlagolásokat, az eredmény 83,51%-ra javult. Ennek magyarázata a 4.3.1.4. pontban leírtak szerint a jobb zajszűrő képességű operátorok, amik a modellezésnél így előtérbe kerülnek.

A 3. CBO konfiguráció, ahol az 1. konfiguráció mellett a 2.4.2.1. pontban leírt CBE-k közül, kettőt definiáltam. Mindkettő ugyanazt a széles normális eloszlással súlyozó CBO-t használja. Az egyik átlagot, a másik szórást számol az egyes változókon számított értékekből. Mindkét CBE-t a sor végén definiáltam, hogy a ShiftTree csak akkor válassza a modellbe, ha éppen az a legjobb választás. A konfigurációval a pontosság 74,59%-ra romlott. A felépített modellben az átlagot számító CBE-t az algoritmus a legfontosabb, 0. szinten választotta, mégis romlott a pontosság. Ez arra utal, hogy a tanítóhalmazban valami olyan speciális szabály érvényes, ami a teszthalmazra már nem, és amit ez az operátor jól meg tud fogni. Azaz a modell túltanult.

A 4. CBO konfiguráció, ami megegyezik a 2. konfiguráció és a két CBE kombinációjával, viszont javulást eredményez a 3. konfigurációhoz képest, de szintén jelentős romlás a 2. konfigurációhoz képest: a pontosság 76,48%. Az ok itt is a túltanulás lehet.

5.2.1.3. Fordított teszt

Az 1. CBO konfiguráció mellett a találati arány 75,56%. Rosszabb, mint a normál teszt során. Ennek az az oka, hogy a teszthalmaz eltérő hosszú idősorokat és általánosabb összefüggéseket tartalmaz, mint a tanítóhalmaz, tehát nehezebb megtanulni.

A 2. CBO konfiguráció itt is javított valamennyit az eredményeken, a 4.3.1.4. pontban leírtak miatt. A pontosság 78,52% lett.

A 3. CBO konfiguráció, ellentétben a normál tesztnél látottakkal, javított az eredményeken, a pontosság 79,26% lett. A javulás oka az, hogy a teszthalmaz általánosabb szabályokat tartalmaz, mint a tanítóhalmaz. Ez egyrészt a méret különbségből ered, hiszen több pont általában általánosabb összefüggéseket ad. Az átlagot számító CBE a modellben itt is magas szinten van: két 2. szintű csomópontban szerepel. De mint látható nem él a 0. szintű speciális szabály, ami a túltanulást eredményezte a normál teszt esetében.

A 4. CBO konfiguráció a fentiek alapján elvárt eredményt adta: javított a 2. és a 3. konfigurációhoz képest is. A pontosság 80,74% lett.

5.2.2. Keresztvalidációs eredmények

A mérés egyik célja az volt, hogy összehasonlítsam egy többváltozós adatsor pontosság és futási idő görbéjét az egyváltozós adatsoroknál kapott eredményekkel. A másik cél az 5.2.1.2. és 5.2.1.3. pontokban tapasztaltak alapján az volt, hogy megnézzem, hogy a rendszer – most specifikusan a feltételállító – komplexitása hogyan hat az egyes görbékre. Ezért 3 különböző komplexitású CBO konfigurációval végeztem el a keresztvalidációt. Az összehasonlíthatóság miatt pontosan ugyanúgy randomizált adatsoron futott mindhárom mérés. A keresztvalidációkat a 4.4.1.1. pontban leírt mérési módszerrel végeztem el.

5.2.2.1. A program konfigurációi

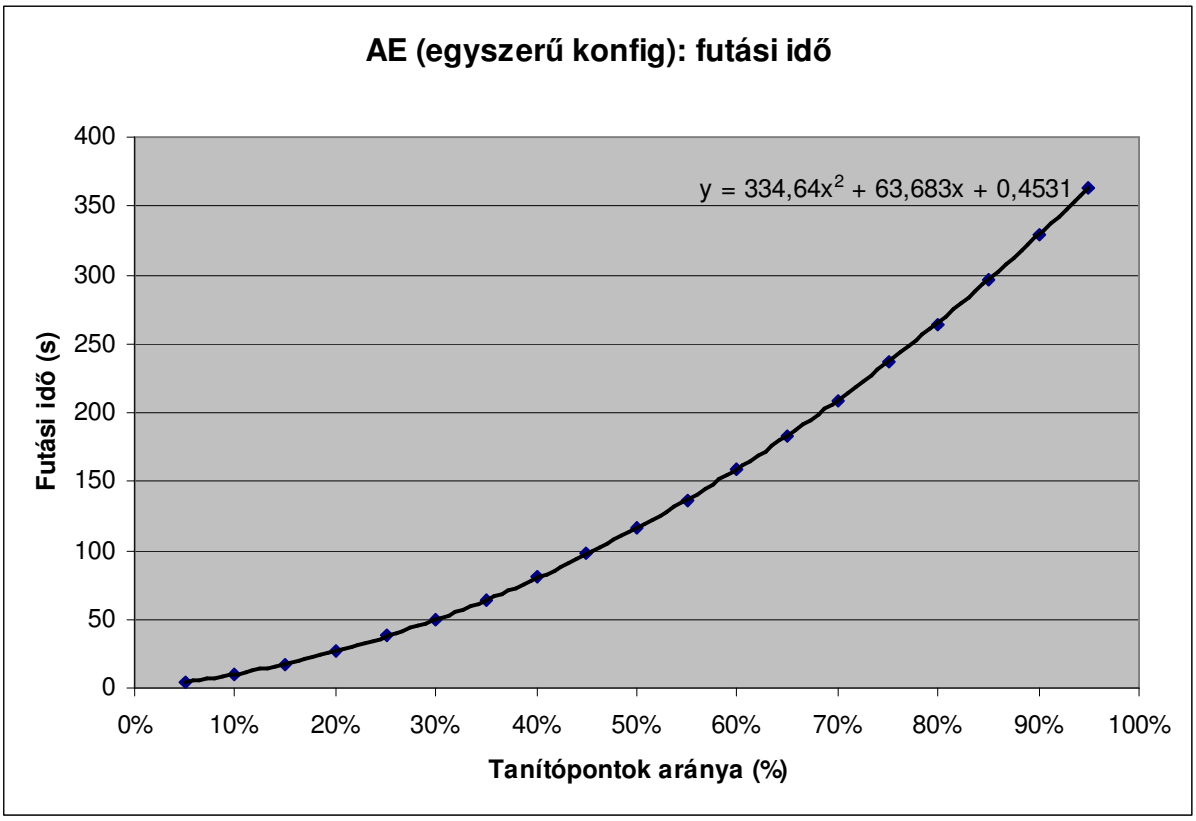
Decider: DEntropy

ESO-k (sorrendben):	NextMax, NextMin, Next(1), Next(2), Next(4), Next(8), Next(12), Next(16), Next(24), Next(32), Max, Min, Prev(1), Prev(2), Prev(4), Prev(8), Prev(12), Prev(16), Prev(24), Prev(32), PrevMax, PrevMin
1. CBO konfiguráció: (egyszerű rendszer)	Simple, Normal(0, 1, 0.05), Exp(1, 0.05), Normal(0, 0.5, 0.01), Exp(0.5, 0.01), Normal(0.5, 4, 0.01), Exp(2, 0.01)
2. CBO konfiguráció (közepes rendszer)	Simple, Normal(0, 1, 0.05), Exp(1, 0.05), Normal(0, 0.5, 0.01), Exp(0.5, 0.01), Normal(0.5, 4, 0.01), Exp(2, 0.01), CBEAverage(Simple), CBEVariance(Simple)
3. CBO konfiguráció: (komplex rendszer)	Simple, Normal(0, 1, 0.05), Exp(1, 0.05), Normal(0, 0.5, 0.01), Exp(0.5, 0.01), Normal(0.5, 4, 0.01), Exp(2, 0.01) , CBEAverage(Simple), CBEVariance(Simple), AVG(5), CBEAverage(AVG(5)), CBEVariance(AVG(5)) , CBEAverage(Normal(0.5, 4, 0.01)), CBEVariance(Normal(0.5, 4, 0.01)), CBEAverage(Exp(2,0.01)), CBEVariance (Exp(2,0.01))

Információnyereség minimális határa: nincs
Szintkorlát: nincs

5.2.2.2. Többváltozós eredmények összehasonlítása az egyváltozósakkal

Az összehasonlításhoz az egyszerű konfigurációt használom, mivel az egyváltozós esetben is ugyanezeket az operátorokat használtam a keresztvalidációnál.



5.2. ábra: Az AE adatsor futási idő görbéje az egyszerű konfiguráció mellett

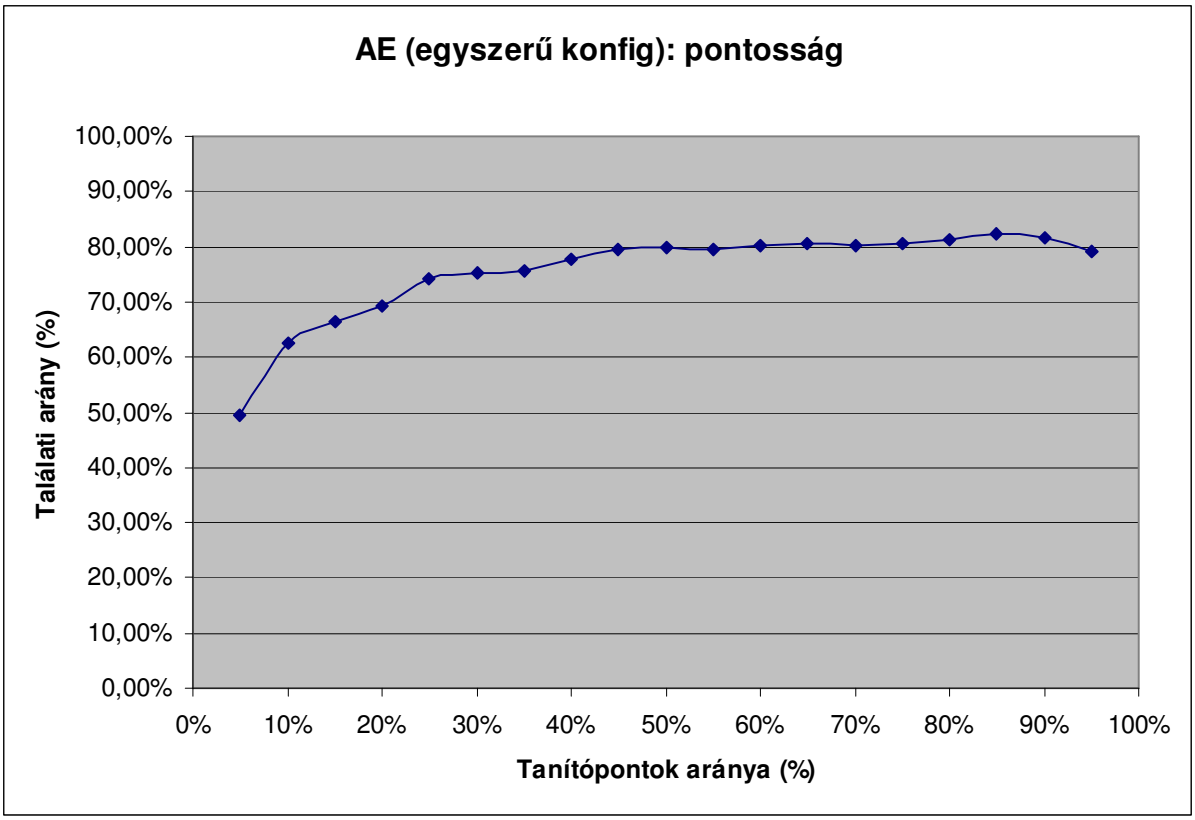
Az 5.1. ábrán látható a futási idő görbéje, amin jól látszik, hogy ebben az esetben is egy négyzetes függvény, csak az együtthatók mások. Az elmélet szerint a futási idő a N_{VR} és N_{VR}^2 közötti szám szorosára növekszik a 3.2. pontban leírtak alapján.

Összehasonlítva a fa és az adatsor mérete szerint leginkább hasonló egyváltozós adatsorral, a synthetic_control-lal, az alábbi 5.3. táblázatot kapjuk.

Adatsor	Tanítópontok száma	Osztályok száma	Nem-levelek száma 95%-on	Nem-levelek száma 50%-on	X^2 együttható	X együttható
AE	640	9	32,35	21	334,64	63,683
Synthetic control	600	6	16,8	13,4	7,059	1,6821

5.3. táblázat: Az AE és a synthetic_contol adatsorok összehasonlítása

Az X^2 együttható, a mi a futási időt a leginkább meghatározza, a többdimenziós esetben 47,41-szerese az egydimenziósnak. Egy azonos méretű adatsor és modell esetén az egyváltozóshoz képest $\frac{6}{22} * 12 * 12 = 39{,}27$ -szoros futási idő növekedést várnánk, mivel a 22 ESO-ból csak 6 darab fut le többször a többváltozós esetben. De a synthetic_control esetében mind a modell, mind az adatsor mérete kisebb, mint az AE adatsor és a hozzá épített modell. Éppen ezért a kapott 47,41-szeres szorzó teljesen elfogadható.



5.2. ábra: Az AE adatsor pontosság görbéje és az osztályozás hibái egyszerű konfigurációnál

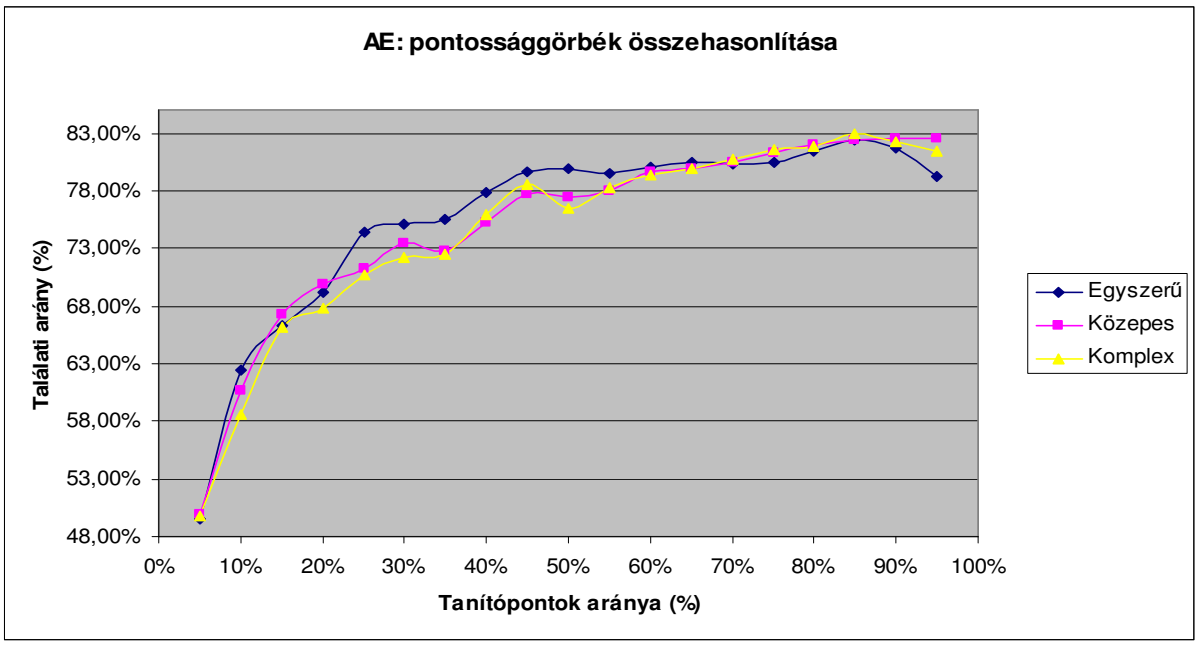
Az 5.2 ábrán látható az AE adatsor pontosság görbéje. A 4.4.5.5.2. pontban leírt kategóriák közül a normál görbék közé tartozik, bár egy kis hullámzás megfigyelhető 80% és 95% között. Az adatsor tulajdonságai is jól illenek a normál görbéket jellemző paraméterek közé.

A görbe 80-82% százalék környékén áll be, ami ennyi osztályt tartalmazó adatsor esetén kiemelkedő (baseline: 18,44%), főleg tanulás-intenzív módszerek esetén. A pontosság az egyszerű tesztek alapján kb. 5%-kal növelhető lenne optimálisabb CBO konfigurációval.

5.2.2.3. Komplexitás hatása a görbékre

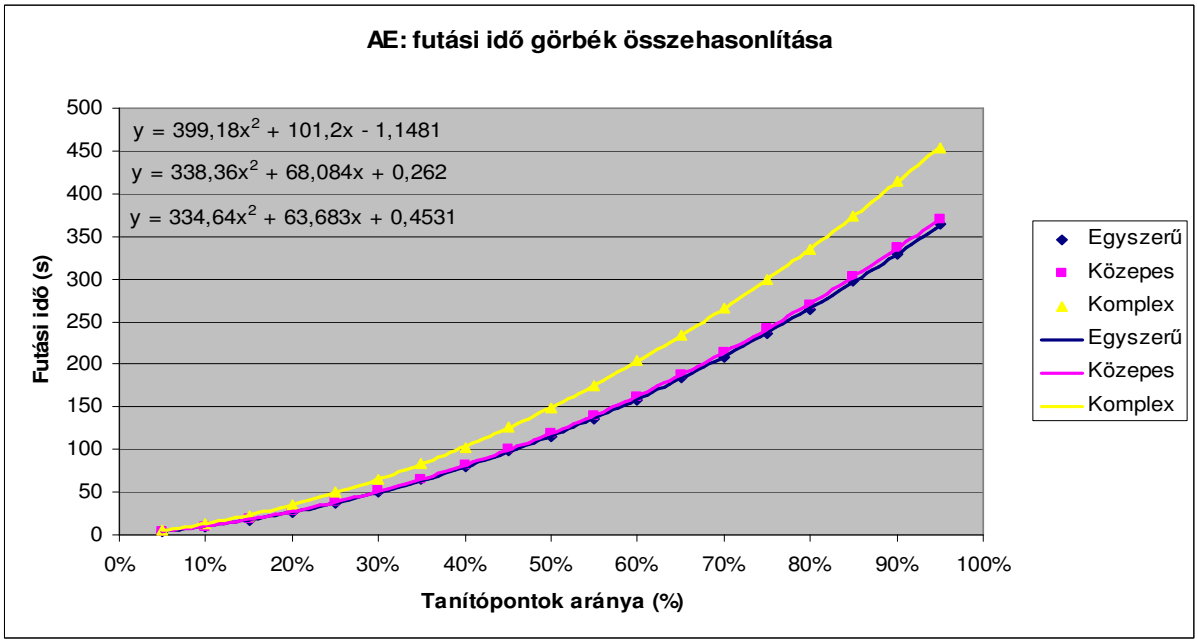
Az elmélet szerint minél komplexebb egy rendszer, annál több memóriával rendelkezik, és annál több, specifikusabb szabályt tud megtanulni. Ezért a komplex rendszerek hajlamosak a túltanulásra. Ezzel szemben az egyszerűbb rendszerek, mivel kevés memóriával rendelkeznek, általánosabb, intuitívabb szabályokat tanulnak meg. Az általános szabályok viszont általában pontatlanabb osztályozást eredményeznek, mint azok a specifikusabb szabályok, amik képesek jobban megfogni az egyes osztályok jellemzőit.

A rendszer komplexitása mellett a tanítóhalmaz mérete is kérdéses, hiszen minél kisebb, annál halmazspecifikusabb és minél nagyobb, annál általánosabb szabályokat tartalmaz az adatsor egészére érvényes szabályokat nézve. Éppen ezért kis tanítóhalmazoknál az egyszerűbb rendszerek általában pontosabbak, hiszen nem tanulják meg az adatsor egészét egyáltalán nem jellemző specifikus szabályokat. Viszont nagyobb adatsorok esetén a komplex rendszerek előnye egyre inkább megjelenik, hiszen az adatsor egészét jellemző szabályok közül többet, bonyolultabbakat meg tud tanulni, mint az egyszerű rendszerek.



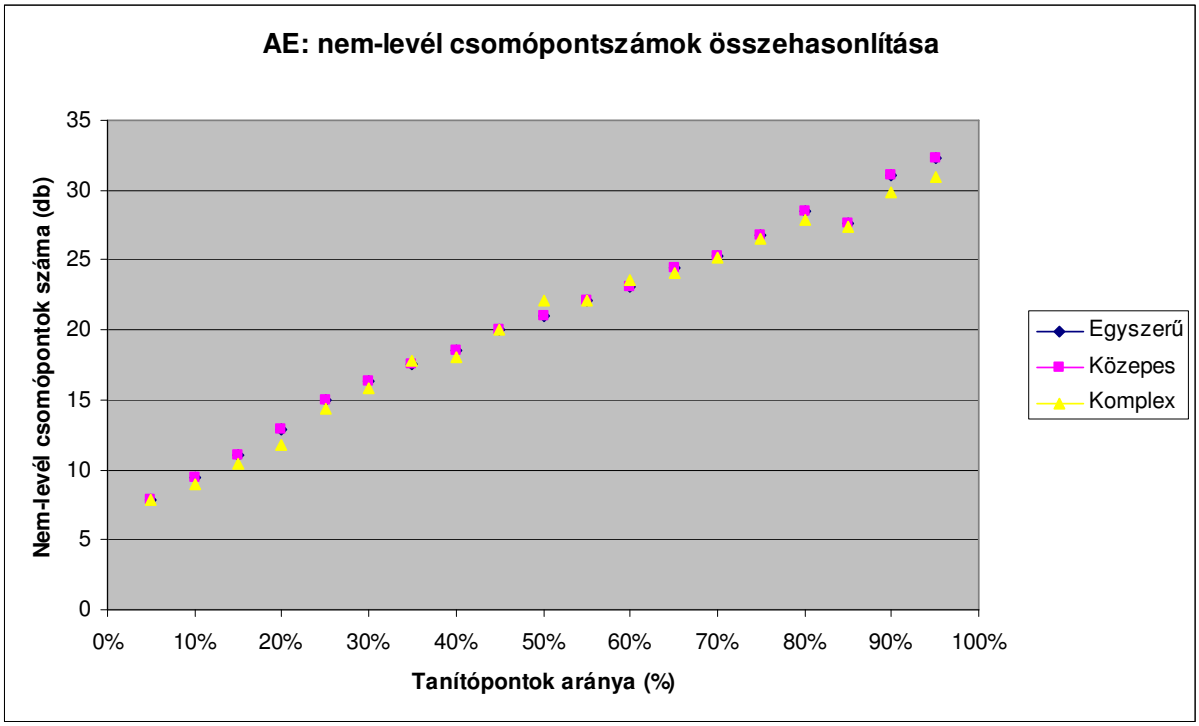
5.3. ábra: Az AE adatsor pontosság görbéinek összehasonlítása az eltérő konfigurációknál

Az 5.3. ábrán látható az 3 eltérő komplexitású konfiguráció által eredményezett pontosság görbe. A görbék teljesen megfelelnek az elméletnek, hiszen jól látható, hogy az egyszerű konfigurációhoz tartozó kék görbe 65%-os arányig a kezdeti zajos helyek kivételével a másik két görbe fölött fut. Ezután a közepes bonyolultságú konfiguráció rózsaszín görbéje van legfölül, szinte a mérések végét jelző 95%-os arányig. Úgy látszik, hogy ez a görbe 83%-nál állna be, pont annyi tanítópont környékén, amennyi 95%-nál van. A komplex konfiguráció sárga görbéjén látszik az, hogy egyrészt nagyon hullámzik, ami annak a következménye, hogy sok specifikus szabályt megtanul, azaz kevés a jó teljesítményéhez a tanítópontok száma. Másrészt az is megfigyelhető, hogy a sárga görbe növekvő trenddel rendelkezik. Azaz ha lenne kétszer ennyi pontunk, akkor megfigyelhető lenne, hogy a legkomplexebb konfiguráció görbéje a tanítópontok számának növekedésével egyre simább lenne, és a rózsaszín görbét lehagyva 84-85% felett beállna.



5.4. ábra: Az AE adatsor futási idő görbéinek összehasonlítása az eltérő konfigurációknál

Az 5.4. ábrán látható a futási idők görbéinek összehasonlítása. Ahogy azt vártam, a komplexebb rendszer több ideig fut, hiszen több operátort kell lefuttatni.



5.5. ábra: Az AE modellek nem-levél csomópontszámainak összehasonlítása

Az 5.5. ábrán látható az, hogy az adott komplexitású konfigurációval készült modell egy adott tanítópont aránynál átlagosan hány nem-levél csomópontot tartalmazott. Jól látható, hogy a komplexebb modellek, ha kevéssel is, de kevesebb nem-levél csomópontot tartalmaznak, mint az egyszerűbbek. Ennek az az oka, hogy a bonyolultabb rendszerek a tanítópontok szempontjából jobb, specifikusabb vágásokat találnak, ezért hamarabb ér el a modell osztályok szempontjából homogén csomópontokat, ahol a modellépítés leáll.

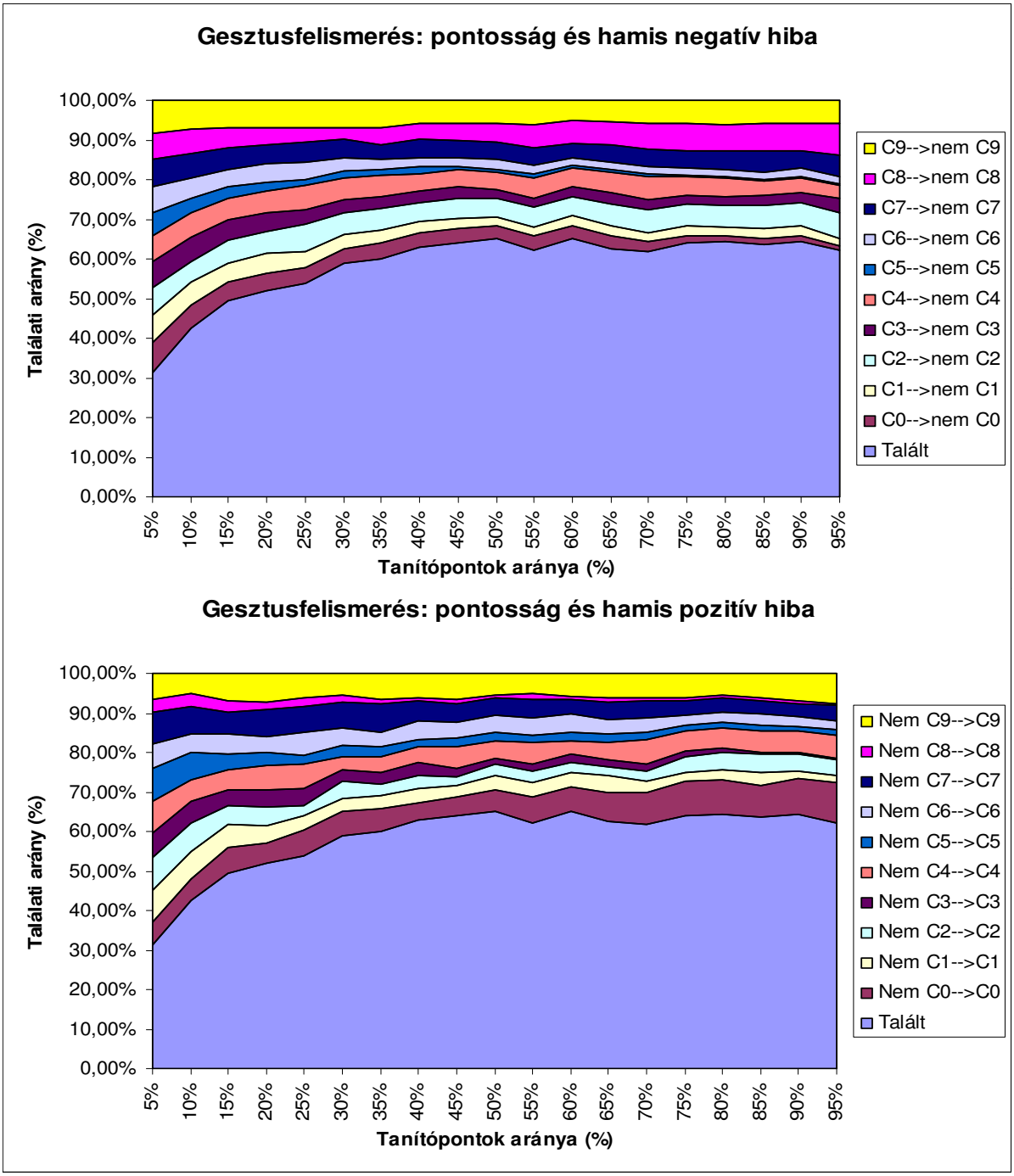
5.3. Gyorsulásmérő adatainak eredményei

Mint ahogyan azt az 5.1.2. pontban említettem, két osztályozási feladat is értelmes ezekkel az adatokkal, ezért mindkettőt kipróbáltam. Ugyanakkor már itt megjegyzem, hogy az adatsor sajnos eléggé kevés adatot tartalmaz. Összesen 502 + 50 zaj idősorunk van, és lényegében 40 + 1 osztály, hiszen az egyes gesztusokat az egyes emberek eltérően csinálják, ez teszi lehetővé az egyes emberek felismerését. Annak ellenére, hogy két felhasználó egy adott gesztusa jobban hasonlít egymásra, mint egy felhasználó két különböző gesztusa, az algoritmust ez a diverzitás elég jelentősen zavarja.

5.3.1. Gesztusfelismerés

Kétféle osztályzást végeztem, az egyiknél 10 osztályba soroltam a 10 gesztust, a másiknál a 10 osztály mellett bevezettem még egyet, ami a zaj-jellegű idősorok osztálya. Mindkét osztályozásnál átlagosan kb. 50 tanítópont állt rendelkezésre osztályonként, ami a 4.4.5.5. pontban leírtak alapján a növekvő vagy hullámzó görbék csoportjába tartozó görbét ad.

5.3.1.1. Gesztusok elkülönítése



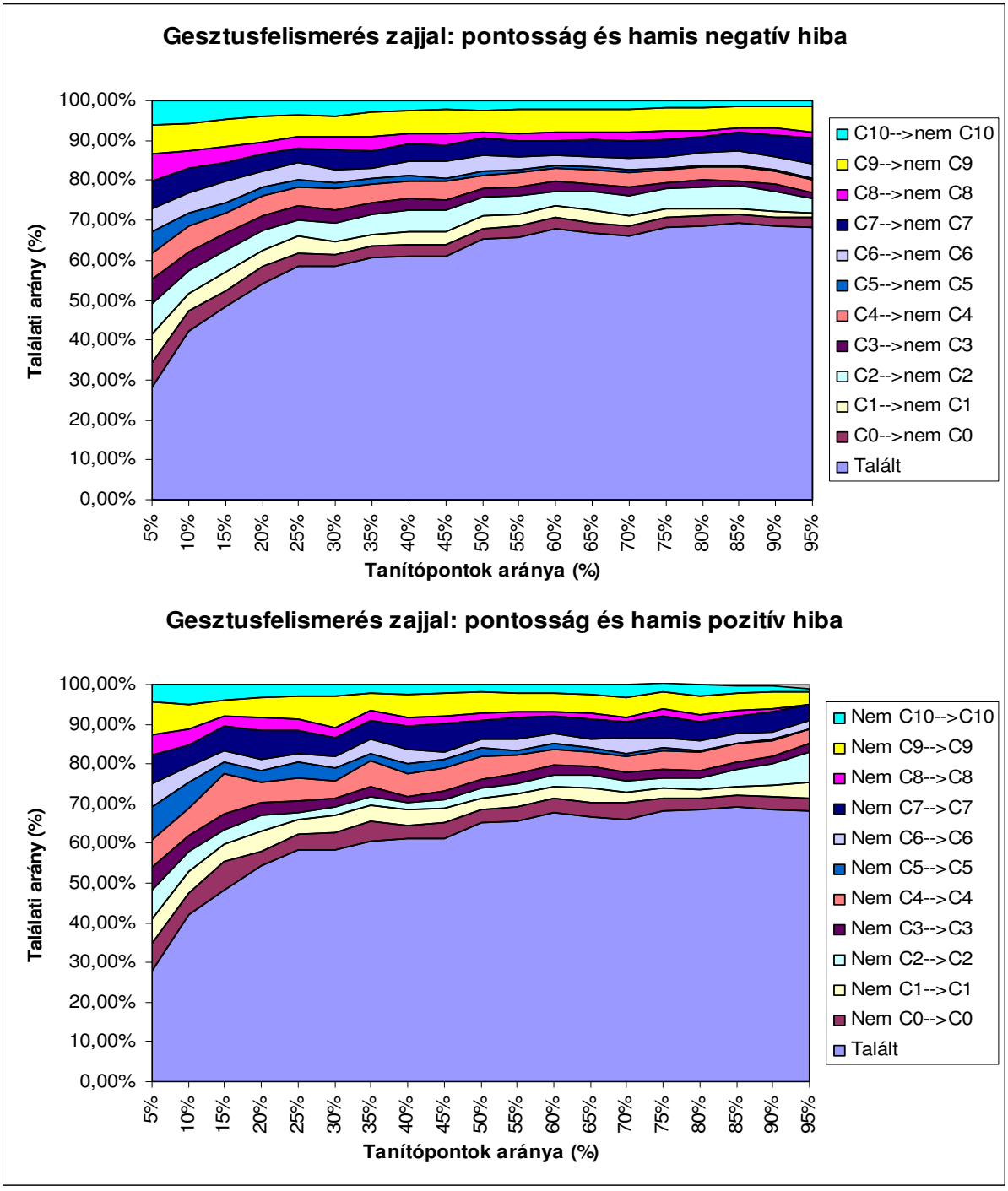
5.6. ábra: A gesztusfelismerés pontossága és a különböző típusú hibák

Az 5.6. ábrán a „Talált” jelmagyarázat mellet lévő világoskék színnel kitöltve látszik a felépített modell pontossága a tanítópontok arányának függvényében. Felette különböző színekkel látszanak a különböző hibák. Az ábra felső részén lévő grafikonon láthatóak azok a hibák, amikor egy adott osztályba tartozó elemet nem az adott osztályba sorol a modell. Az alsó grafikonon szerepelnek azok a hibák, amikor egy olyan pontot sorolok egy adott osztályba, ami nem oda tartozik.

Az ábrán jól látszik, hogy az eltalált idősorok határa (pontosság görbe) erőteljesen hullámzik a kevés tanítópont miatt. Érdekes még megfigyelni, hogy a C5 osztályba tartozó pontokat szinte mindig eltalálja a modell (felső grafikonon vékony a „C5→nem C5”-höz tartozó

sötétkék sáv). Ezzel szemben a C8-hoz tartozó pontokat sokszor máshová sorolja (rózsaszín sáv a felső grafikonon). A C5 olyan gesztus, ahol a telefont vízszintesen oda-vissza „megrázza” a felhasználó. Ez könnyen elkülöníthető, hiszen csak a vízszintes irányhoz tartozó változó tér el jelentősen a 0-tól. A C8, egy hullámzást imitáló mozgás, aminek az idősora nehezebben ismerhető fel. Szintén érdekes az alsó grafikonon megnézni, hogy a már említett C8 osztályba szinte alig sorol be olyanokat, akik nem oda valók, és a C9-be sorolja a legtöbb olyat, ami nem is oda tartozik.

5.3.1.2. Gesztusok és zaj elkülönítése



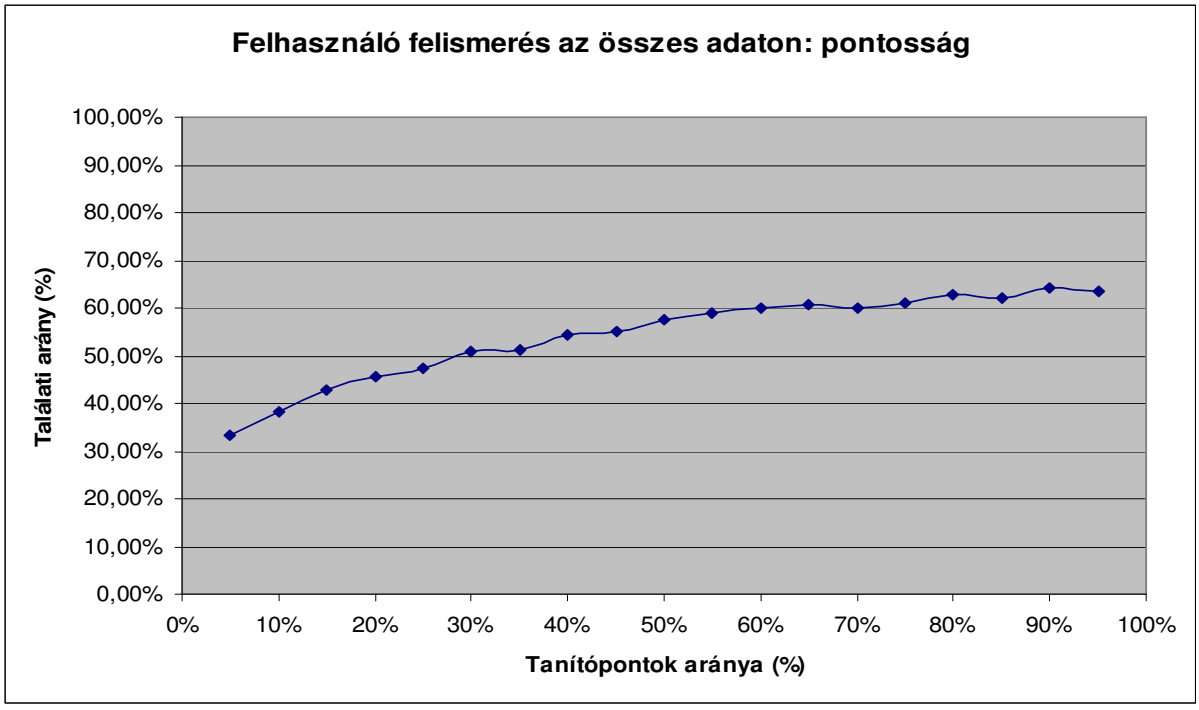
5.6. ábra: A gesztus- és zajfelismerés pontossága és a különböző típusú hibák

Az 5.6. ábra mutatja a gesztusok felismerését abban az esetben, ha a zaj-jellegű adatokat is bevettem az osztályozásba, mint 11. osztályt. Látható, hogy az előző osztályozáshoz képest a pontosság kb. 5%-kal javult. Ennek az az oka, hogy a zajt az algoritmus eléggé jól el tudja különíteni. Ezt bizonyítja a vékony ciánkék sáv mindkét grafikonon: kevés zajt sorol be más osztályba és kevés olyan idősort nyilvánít zajnak, ami nem az.

5.3.2. Felhasználó felismerése

A felhasználók felismerése kétféle módon történhet. Az egyik, hogy az algoritmus az összes gesztust veszi, és így megpróbálja elkülöníteni a felhasználókat. A másik, hogy gesztusonként próbálja elkülöníteni őket. Az előbbi probléma nagyon nehéz, hiszen két felhasználó által leírt két ugyanolyan gesztus nagyon hasonló, míg egy felhasználó által leírt két különböző gesztus nagyon eltérő. Az osztályozást az is megnehezíti, hogy nem a gyakorlatban előforduló adatokról van szó, mivel a telefont minden esetben függőleges helyzetből indították, és úgy is fejezték be a mozdulatot. Tehát az algoritmus annak alapján se tudja felismeri a felhasználót, hogy észreveszi, hogy mondjuk az adott személynek úgy szokása befejezni a mozdulatot, hogy zsebre vágja a telefont. Mindkét esetben tovább nehezíti a dolgot az, hogy az adatok között egy felhasználó kétszer annyiszor szerepel, mint a többiek. Ezen kívül a kevés számú idősor is hátráltatja a ShiftTree-t.

5.3.2.1. Összesített felismerés

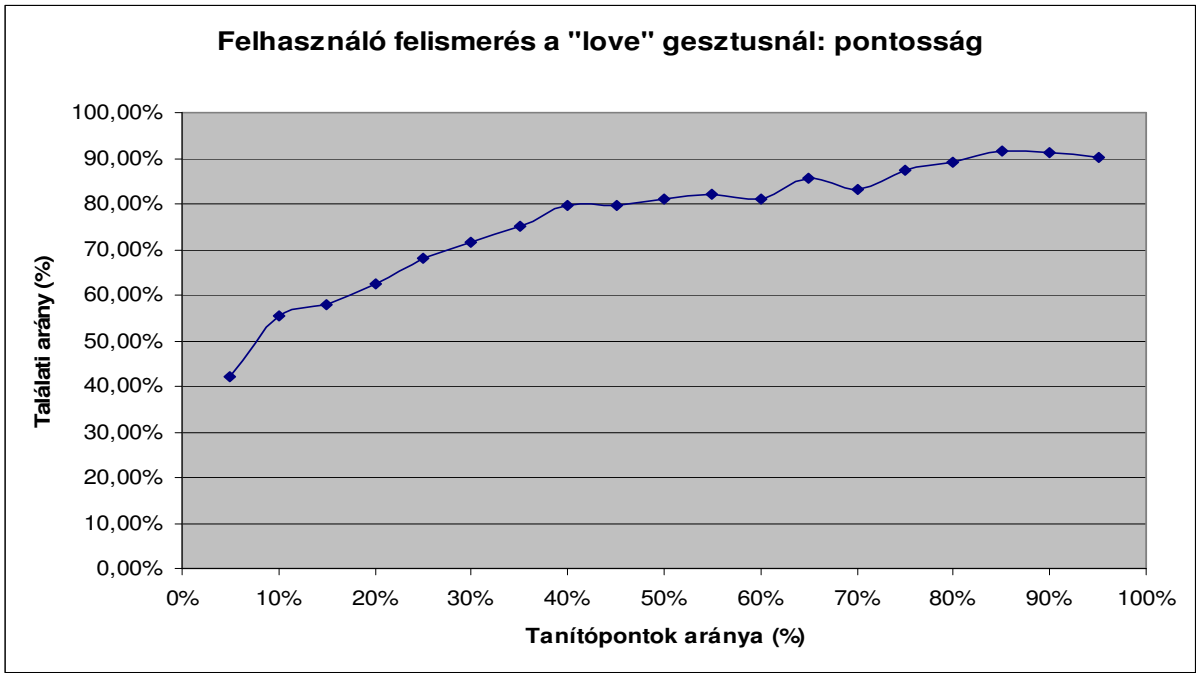


5.7. ábra: A felhasználó felismerése a teljes adatsoron

Az 5.7. ábrán látható, hogy milyen pontossággörbét produkált az összesített felhasználó felismerés modellje. Ahhoz képest, hogy mennyire bonyolult a feladat, és mennyi minden nehezíti az algoritmus dolgát, az 55%-65% közötti pontosság egész jó eredmény. Ráadásul 60%-os tanítópont aránytól 60%-os pontosság felett vagyunk (baseline: 39,84%).

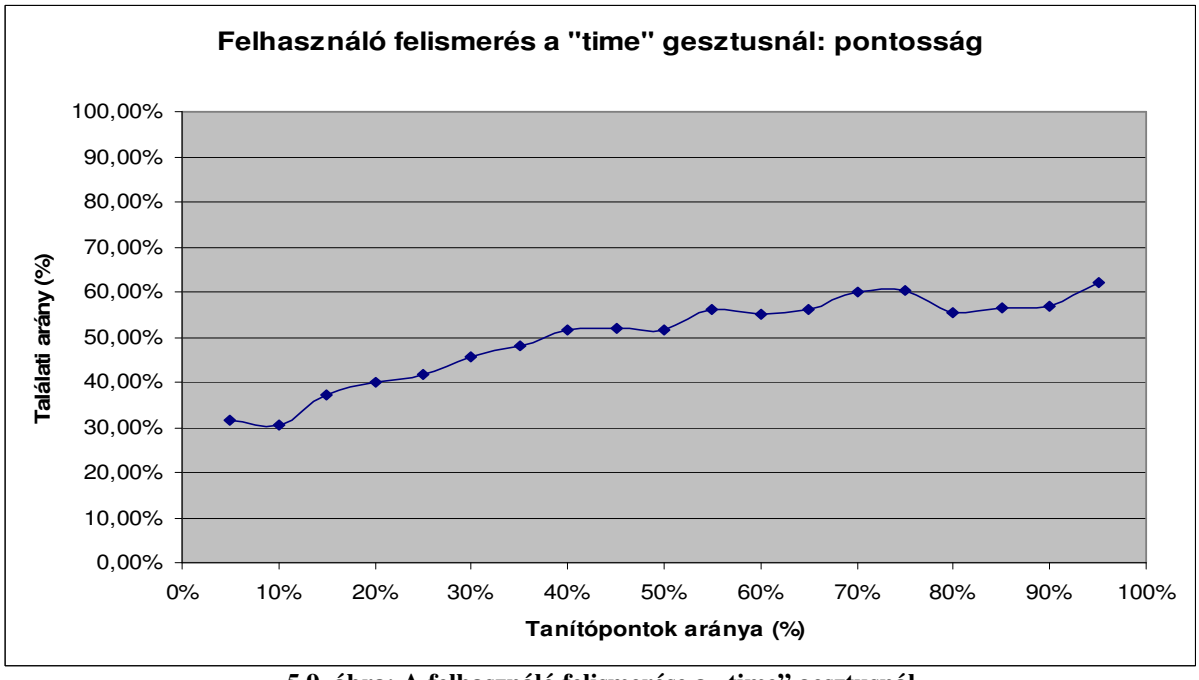
5.3.2.2. Gesztusonkénti felismerés

A gesztusonkénti felhasználó felismerés eléggé eltérő eredményeket hozott gesztusonként. A legjobbat és a legrosszabbat elemzem lentebb.



5.8. ábra: A felhasználó felismerése a „love” gesztusnál

Az 5.8. ábrán látható a legjobban sikerült felismerés, ami a „love” nevezetű gesztusnál történt. A baseline itt 39,22%, amitől a 80%-90%-os eredmény jelentősen jobb. Elég tanítópont esetén nem kizárt, hogy 95%-ig feljutna a görbe. A kiemelkedő pontosság oka, hogy a gesztus egy bonyolultabb mozgás, egy levegőbe rajzolt szív. Érthető, hogy azért sikerült ilyen jól az osztályozás, mert ez egy bonyolult mozdulat, amit a legtöbb ember eltérően hajt végre.



5.9. ábra: A felhasználó felismerése a „time” gesztusnál

Az 5.9. ábrán látható a leggyengébb osztályozási eredmény, amit a „time” gesztusnál kaptunk, ami két kis körzés az óramutató járásával megegyező irányba. Ez azért van, mert a gesztus egy viszonylag egyszerű mozdulat, ami ráadásul egyszerű függvényeket eredményez (sinus jellegű függvény), tehát a legtöbb ember függvénye hasonló alakú, csak az értékeikben térnek el. Az ilyen eltérés viszont egy ember két gesztusa között is érvényes, tehát a minták „összekeverednek”, ami miatt a ShiftTree nem tudja őket hatékonyan szétválasztani.

6. Optimalizációs módszerek

A 4.2. pontban láthattuk, hogy az optimalizálatlan algoritmus más módszerekkel összehasonlítva milyen eredményeket ér el. Azt állapítottam meg, hogy hasonlóan nem optimalizált algoritmusokkal összehasonlítva az élmezőnybe kerül, viszont az optimalizált algoritmusokkal szemben csak a középmezőnybe jut.

Ebben a fejezetben a ShiftTree optimalizálásáról lesz szó. Egy – külön ehhez a módszerhez fejlesztett – automatikus optimalizációs eljárással, a többszörös modellezéssel kezdem, majd áttérek a hagyományos döntési fáknál is használt utónyesés ShiftTree specifikus verziójának ismertetésére.

6.1. Többszörös modellezés

A többszörös modellezés egy olyan automatikus optimalizációs eljárás, ami kifejezettem a ShiftTree algoritmushoz készült, annak sajátosságait használja fel. A 4.3.1.4. pontban, a CBF adatsorhoz épített modell elemzésének végén láthattuk, hogy az operátorok definiálásának a sorrendje jelentős hatással lehet az osztályozás pontosságára. Az említett pontban a CBO-k sorrendjét vizsgáltam, de ugyanez elmondható az ESO-k sorrendjéről is.

A sorrend hatása onnan adódik, hogy egy csomópontban több ESO-CBO párhoz a decider adhatja ugyanazt a minimális jóságértéket, azaz a tanítóhalmaz szempontjából több vágás is lehet optimális. Ebben az esetben azt az ESO-CBO párt fogja feljegyezni a modellbe, amelyik az első minimális jóságértékű vágáshoz tartozik. Ez a vágás viszont nem biztos, hogy a teszthalmazon a legjobb eredményt hozza. Ez általában abból adódik, hogy a tanítóhalmaz mérete kisebb a teszthalmazénál, ezért az előbbiben nincs annyi információ, ami alapján eldönthető lenne, hogy a teljes adatsort melyik vágás jellemzi a leginkább.

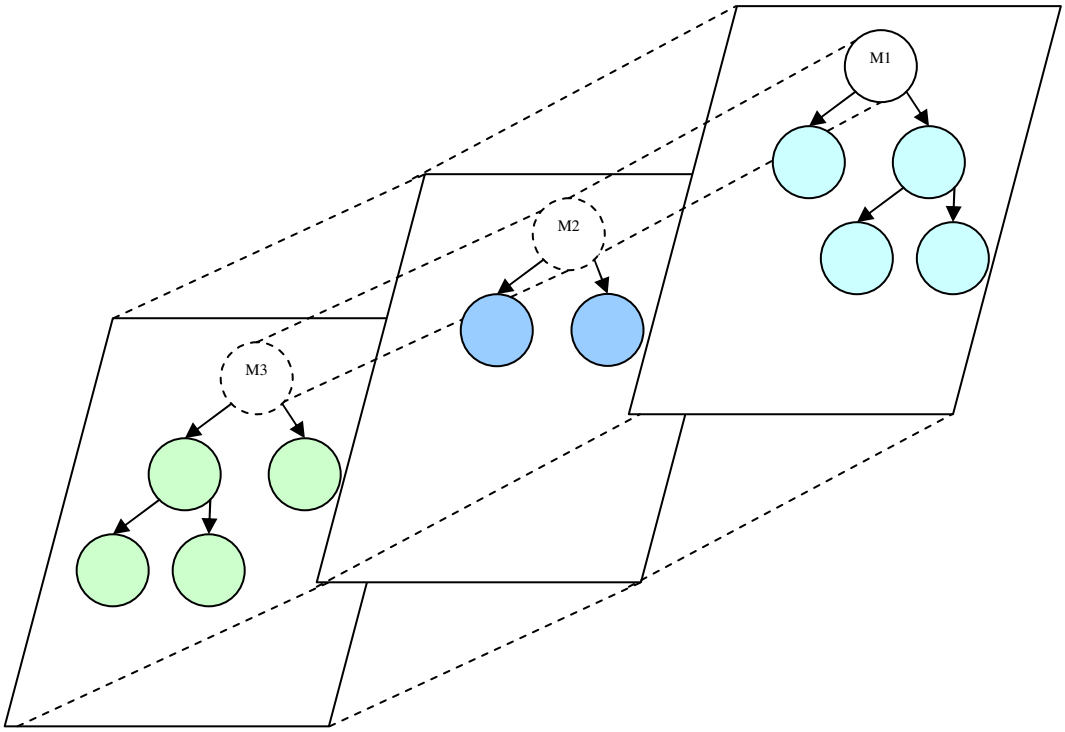
Egy megoldás lehetne, hogy minden lehetséges sorrenddel modellt építünk úgy, hogy az ESO-kat és a CBO-kat ciklikusan shifteljük minden modellépítés között. Ennek a módszernek két hátránya van. Az első, hogy sokszor feleslegesen építünk új modellt, mert egy olyan sorrendet definiáltunk, ami egy előző modellel megegyező eredményt ad. A másik, hogy nem eredményez teljes optimalizációt, mivel lehet, hogy egy csomópontban az egyik sorrend szerinti első vágás adja majd a legjobb eredményt a teszthalmaz szempontjából, addig egy másik csomópontban más sorrendezés bizonyul a legjobbnak.

A többszörös modellezés ezeket a hátrányokat küszöböli ki. Ebben a pontban először az eljárás lényegét ismertetem, majd kitérek a módszerrel elért eredményekre, különös tekintettel az optimalizálatlan algoritmussal és a 4.2. pontban megvizsgált más algoritmusokkal való összehasonlításra.

6.1.1. Az eljárás ismertetése

Az eljárás lényege – ahogy azt a neve is sugallja – hogy több modellt építünk. A 2. fejezetben láthattuk, hogy a modelleket csomópontonként tartja nyílván a ShiftTree, ezért ez a módszer a csomópontokban megtalálható modelleket többszörözi, és fogja őket egy többszörös modellbe (MultiModel).

Csomópontonként a decider által legjobbnak ítélt vágások bekerülnek egy speciális modellbe, ami a korábbi modellhez képest annyival bővült, hogy a vágáshoz tartozó két gyermek csomópontra mutató pointer is benne található meg, nem pedig a modellt tartalmazó csomópontban. Ezzel a módszerrel összerendelődik a vágás, és a hozzá tartozó csomópontok, ami a későbbiekben számtalan lehetséges hibaforrást megszűntet. A 6.1. ábra egy egyszerű többszörös modellezés által épített fát ábrázol szemléletesen.



6.1. ábra: Többszörös modellezés ábrázolása

Az ábrán felül lévő csomópontban 3 különböző vágás bizonyult a legjobbnak, ezért három különböző modell készült (M1, M2, M3). Mindegyik modellhez tartozik két gyermek csomópont, amik a modellekben megtalálható vágással jönnek létre. Ezt szemléletesen úgy ábrázolhatjuk, mintha a csomópont több síkra vetülne, és a különböző síkokon különböző módon történne meg a vágás. Egy ilyen síkon egy ShiftTree alakul ki, és minden egyes többszörös vágás $N_v - 1$ síkot hoz létre, ha N_v a csomópontban található legjobb vágások (modellek) száma. Egy síkon megjelenő gyermek csomópontoknak (tehát a vágást okozó csomópontot kivéve az összes csomópontnak) semmi köze nincs a többi síkon lévő gyermekekhez. A vágást okozó csomópontok viszont nincsenek megtöbbszörözve (ezért is jelöltem őket az ábrán szaggatott vonallal), hanem a fenti leírásnak megfelelően több modellt tartalmaznak, és minden modelljük tartalmaz pointereket két gyermek csomópontra.

Az ábra csak egy több modellel rendelkező csomópontot ábrázol, hogy átlátható maradjon. De a módszer lényege, hogy akár a régi, akár az új síkon lévő gyermekcsomópontok szintén tartalmazhatnak több modellt, és így szintén létrehozhatnak új síkokat, mint ahogy ez a módszer leírásában sem volt csupán egy csomópontra korlátozva.

Az osztályozás egy ilyen többszörös fán úgy zajlik, hogy egy csomópontban rekurzívan megnézi az algoritmus, hogy melyik modelljéhez tartozó gyermek csomópontok adják a legjobb találati arányt egy adott teszthalmazra, ezt a számot visszaadja, és a legjobb modellt csomópontonként megjelöli. Ezzel a módszerrel a többszörös fából kiválasztható egy fa, ami a fák közül a legjobb eredményt adja az adathalmazra. Ez a modell akár egy egyszerű ShiftTree-be is átmenthető. Később ez a kiválasztott modell használható az osztályozási feladatra ismeretlen osztályváltozójú adathalmazok esetén.

6.1.1.1. Megszorítások

A módszer egyetlen hátránya a megnövekedett futási idő és memóriaigény, hiszen ha sokszor sokfelé ágazunk, akkor a modellek száma exponenciálisan nő. Éppen ezért megszorításokat kell bevezetni annak érdekében, hogy a futási idő és a memóriahasználat elfogadható legyen.

Az egyik legegyszerűbb, mégis hatékony korlát az, hogy ha megszabjuk, hogy mennyi lehet a síkok (az összes különböző fa) száma. Ha elértük a megszabott határt, akkor nem állunk le a fák építésével, csak az ezután következő csomópontokban maximum egy modellt (vágást) engedélyezünk. Ilyenkor viszont figyelni kell arra, hogy a csomópontokat ne a szokásos inorder bejárás szerint fejtsük ki, mivel a magasabb szinten lévő vágások fontosabbak, mint az alacsonyabb szinten lévők (nagyobb a hatásuk a modellek különbözőségére). Inorder bejárás szerint egy ágat fejtenénk ki ameddig lehet, és könnyen előfordulhatna, hogy elérjük a modellkorlátot, ami miatt a többi ágon már csak egyszeres modelleket használhatnánk. A probléma megoldása egy FIFO sor, amibe a létrejött gyermek csomópontokat rakjuk, és mindig az elől lévő csomópontot fejtjük ki.

Bizonyos vágások nagyon eltérő méretű (eltérő számú tanítóponttal rendelkező) gyermek csomópontokat eredményeznek. A nagyobb csomópontok vágásai fontosabbak, a modellek különbözősége szempontjából. Ráadásul a nagyon kevés tanítóponttal rendelkező csomópontok komoly problémát jelentenek, mivel szinte az összes ESO-CBO párral ugyanolyan jó vágásokat kaphatunk. Ezzel akár 100+ modell jöhet létre egy csomópontban, amihez 200+ gyermekcsomópont tartozik, amiket esetleg tovább vághatunk. Ezzel egyrészt gyorsan elérjük az engedélyezett síkok számát jelentéktelen vágásokkal, másrészt jelentős többletmunkát adunk a rendszernek. Éppen ezért célszerű bevezetni, hogy az egy adott elemszámnál kevesebb tanítópontot tartalmazó csomópontok nem hozhatnak létre új síkot, azaz maximum csak egy modellt tartalmazhatnak.

6.1.2. Eredmények az eljárással

A teszteléshez a 4.2. pontban is használt benchmark adatsorokat használtam. A tanítóhalmazon tanítottam az algoritmust, a teszthalmazon mértem le a legjobb modell találati arányát, azaz a teszthalmazra optimalizáltam, mivel pontosan ilyen eredmények állnak rendelkezésemre más algoritmusokkal. A program beállítása a szokásos 4.2.1. pont alatti beállítás volt. A 4.2.1. ponthoz hasonlóan itt is megvizsgáltam a 11 decidert és a 8 féle mélységkorlátot (3, 5, 7, 9, 11, 13, 15, nincs). Viszont nem foglalkoztam a minimális információnyereségi határral. A síkok számát 25000-ben korlátoztam, és bizonyos adatsoroknál adott decider mellett (amiknél szükség volt erre) 4 és 10 között határoztam meg a tanítópontok minimális számát, aminél még a többszörös modellezés engedélyezett.

6.1.2.2. Jóságértékek és a többszörös modellezés

A tapasztalatok azt mutatják, hogy a többszörös modellezés – a fent említett feltételekkel – nem minden jóságérték mellett hatékony. Pontosabban a súlyozott divergenciafüggvény $\lambda \geq 1$ paraméter esetén bizonyos adatsoroknál kifejezetten hosszú futási időt eredményez. Ennek az az oka, hogy ez a jóságérték azokat a vágásokat részesíti előnyben, ahol a gyermekekbe kerülő tanítópontok szám közel azonos, és csak másodlagos szempontként veszi figyelembe, hogy a gyermekek minél homogénebbek legyenek. Ezért az olyan adatsoroknál, ahol az osztályok elkülönítése a csomópontban nem egyértelmű (a legtöbb adatsor ilyen), vagy ahol az egyes osztályok jelentősen eltérő számmal kerülnek be a csomópontba, ott nagyon sok, 70+ vágást talál optimálisnak, és szinte azonos méretű gyermekeket generál. Így aztán egyik gyermek csomópontot sem zárhatjuk ki, ráadásul azok nem eléggé homogének, ezért bennük is sok modell jön létre. Ez a probléma nem jelentős, mivel általában azoknál az adatsoroknál jelentkezik, amik amúgy sem adtak jó eredményt ezzel a jóságértékkel. Ugyanakkor kezelhető is, ha egy homogenitási korlátot állítunk a többszörös modell létrehozásához.

6.1.2.3. Összehasonlítás

A 6.1. táblázat mutatja az eredeti és a többszörös modellt használó ShiftTree eredményeit. A sorok az adatsoron elért legjobb eredményt mutatják (a 11 decider és a 8 féle mélységkorláttal elértek közül). A javulás oszlopban ábrázoltam azt, hogy a többszörös modellezés mennyivel pontosabb, mint az egyszeres modellezés. Látható, hogy minden esetben jobb, vagy egyenlő az eredmény. Ez nem is meglepő, hiszen a többszörös modell mindig tartalmazza az eredeti egyszeres modellt is, ami miatt rosszabb nem lehet.

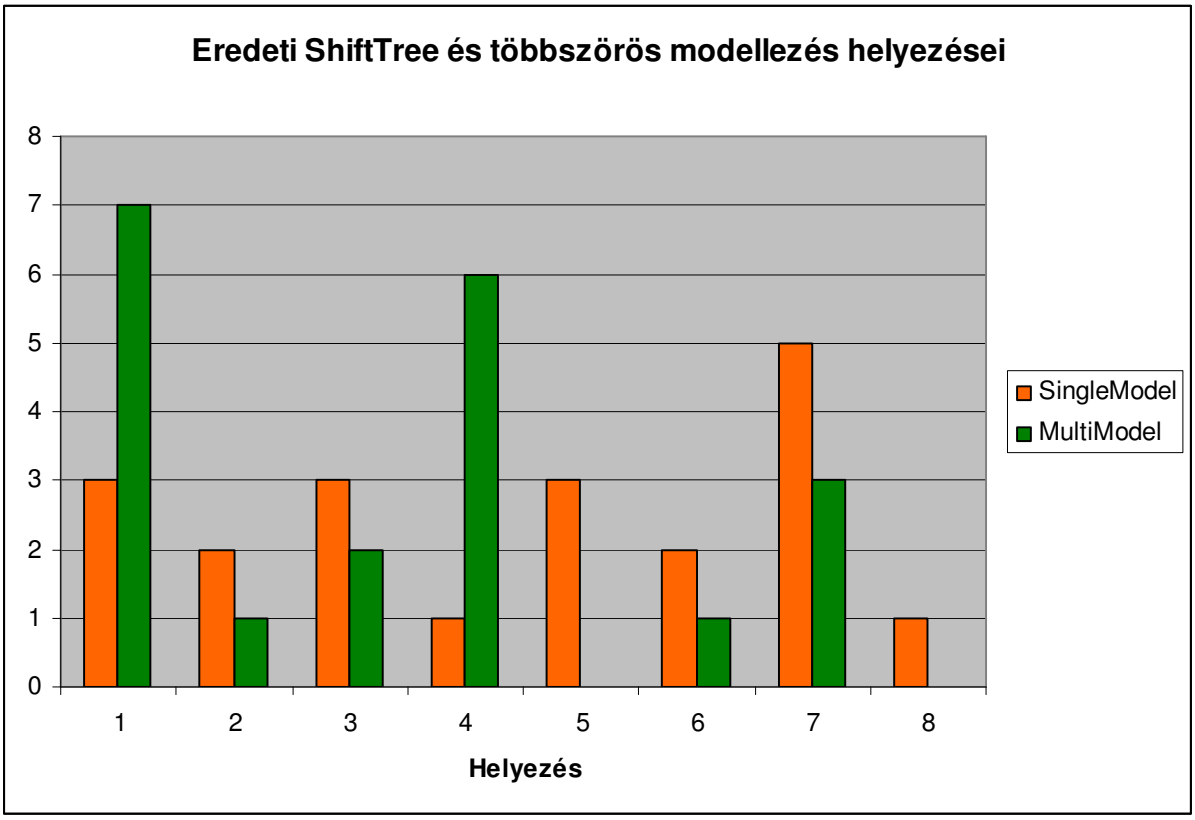
A javulás mértéke adatsoronként jelentősen eltér. Jól látható, hogy a kisebb tanítóhalmazzal rendelkező adatsorok, mint amilyen a Beef vagy a Coffee, esetében jelentős +10-20% körüli növekedés tapasztalható, mivel a tanítóhalmaz kevés információt hordoz. A nagyobb tanítóhalmazzal rendelkező adatsorok, mint amilyen a Yoga vagy a Wafer, esetében a javulás nem számottevő, mivel a tanítóhalmaz eredetileg is elég információval rendelkezett az adatsor pontos leírásához, azaz kevesebbet kell próbálkoznunk, a több modell inkább alsóbb szinteken jelenik meg.

Az alsó két sor összesített eredményeket tartalmaz. A felső sor egy egyszerű átlag, az alsó pedig a teszthalmazok méretével súlyozott átlag. Ez utóbbi lényegében azt jelenti, hogy az összes teszthalmazon eltalált pontok száma hogyan viszonyul az összes teszthalmaz méretének összegéhez. Mivel a nagyobb adatsorokon eleve pontosabb volt az algoritmus (tanulás-intenzív), ezért a súlyozott átlagnál kisebb javulás látható, mint az egyszerű átlagnál.

	Eredeti ShiftTree	Többszörös modellezés	Javulás
50Words	39,78%	43,96%	4,18%
Adiac	52,94%	52,94%	0,00%
Beef	53,33%	63,33%	10,00%
CBF	89,22%	94,67%	5,44%
Coffee	67,86%	85,71%	17,86%
ECG200	82,00%	88,00%	6,00%
FaceAll	61,24%	70,24%	8,99%
FaceFour	71,59%	84,09%	12,50%
Fish	66,29%	69,71%	3,43%
GunPoint	79,33%	86,00%	6,67%
Lighting2	77,05%	83,61%	6,56%
Lighting7	63,01%	72,60%	9,59%
OliveOil	80,00%	83,33%	3,33%
OSULeaf	56,20%	58,26%	2,07%
SwedishLeaf	71,20%	74,56%	3,36%
SyntheticControl	93,67%	96,00%	2,33%
Trace	100,00%	100,00%	0,00%
TwoPatterns	94,18%	94,93%	0,75%
Wafer	98,25%	98,72%	0,47%
Yoga	74,37%	74,93%	0,57%
Átlagosan	*73,58%*	*78,78%*	*5,20%*
Súlyozva átlagosan	*84,80%*	*86,85%*	*2,05%*

6.1. táblázat: Többszörös modellezés és az eredeti algoritmus összehasonlítása

A 6.2. ábrán látható hisztogram a két módszert hasonlítja össze a 4.2. pontban is használt hét algoritmussal, azok optimalizált eredményeivel. Az egyes oszlopok azt jelzik, hogy a kettő közül valamelyik módszert a 7 másik algoritmussal összehasonlítva hány esetben ért el első, második, …, nyolcadik helyezést a 20 adatsor esetén.



6.2. ábra: Többszörös és egyszeres modellezés összehasonlítása más algoritmusokkal

Látható, hogy a zölddel jelölt többszörös modellezés hisztogramja balra tolódik, a narancssárga egyszeres modellezésé pedig jobbra, azaz amíg az optimalizálatlan ShiftTree az optimalizált algoritmusok között valahol a középmezőnyben helyezkedik el, addig a többszörös modellezéssel az élmezőnybe került.

Az ábra mellett egy fontos mérőszám az, hogy az átlagos és a súlyozva átlagolt pontosságok tekintetében melyik módszer hányadik helyezést ért el a többi algoritmussal összehasonlítva. Ez látható a 6.2. táblázatban.

	HELYEZÉS	
	Eredeti ShiftTree	**Többszörös modellezés**
Átlagosan	*5*	*2*
Súlyozva átlagosan	*3*	*2*

6.2. táblázat: Többszörös modellezés és az eredeti algoritmus átlagos pontosságának helye

A többszörös modellezés, mint látható, jelentősen feljavítja az eredményeket. A módszer teljesen automatikus, emberi beavatkozást csak a korlátozások beállításánál igényel. Az viszont a legtöbb jóságértéknél a legtöbb adatsoron tág határok között mozoghat.

6.2. Egy egyszerű utónyesési eljárás

A döntési fákat szokás elő- és/vagy utó nyesni. Az előnyesés azt jelenti, hogy a fa építése közben eldöntjük, hogy bizonyos ágakat nem fejtünk ki, az ott kifejthető csomópontokat „lenyessük”. Az utónyesés a már felépült fáról vág le ágakat valamilyen elv alapján.

Az előnyesés leginkább a futási időt csökkenti azzal, hogy a felesleges ágak kifejtését megakadályozza. Emellett a túltanulás ellen is véd valamennyire. Az utónyesés a futási időt nem csökkenti, de hatékony védelmet biztosít a túltanulás ellen.

Ebben a pontban bemutatok egy nagyon egyszerű utónyesési eljárást, amit a túltanulás megakadályozására építettem bele az algoritmusban. Majd megmutatom, hogy ez hogyan hat a pontosságra egyszerű modellezés és többszörös modellezés esetén.

6.2.1. Az eljárás

Az eljáráshoz alapvetően három idősor halmazra van szükség. Kell egy tanítóhalmaz, egy teszthalmaz és egy validáló halmaz. A tanítóhalmaz alapján felépítünk egy fát az eddigi módszerek valamelyikével.

Ha elkészült a fa, akkor a teszthalmazt elkezdjük a fával osztályozni. Minden csomópontban rekurzívan megnézzük, hogy az odakerült pontok közül akkor osztályoznánk-e többet helyesen, ha a csomópontban megállnánk, vagy ha a gyerekeknek továbbadnánk ezeket a pontokat a felépített modell szerint szétvágva. Azaz megnézzük, hogy a tanítóminta által épített fának mi az a részfája, ami a teszthalmazra a leginkább illeszkedik. Ezt a részfát megjelöljük, és ez lesz a végleges modell, amivel a validáló halmazt osztályozva megkapjuk a modellünk tényleges pontosságát.

A módszer a többszörös modellezéssel készült fákra is ugyanígy alkalmazható. Csak ebben az esetben nem csak a két gyermek csomópontra kell megnézni az eredményt és azt összehasonlítani magával a csomóponttal, hanem az összes modellben szereplő gyermeket meg kell vizsgálni.

A módszer véd mind a tanító, mind a teszthalmaz túltanulásától, hiszen a teszthalmaz nem szól bele a tanításba, a tanítóhalmaz meg a vágásba. Tehát pont azokat az agákat vágjuk le, amik nem közösek a tanító és a teszthalmazban, azaz amik speciális szabályokat tartalmaznak.

6.2.1.1. Alkalmazás két halmaz esetén

A legtöbb adatsor esetén két adathalmaz áll rendelkezésemre. Attól függően, hogy mi a feladat, kétféle képen lehet alkalmazni a fent leírt módszert.

Ha egy adott teszt adathalmazra szeretnék optimalizálni, mint ahogy a benchmark adatsoron futtatott többi algoritmus is ezt tette, akkor a teszthalmazt tekintem teszt- és validáló halmaznak is. Ezáltal azt az eredményt kapom meg, hogy a tanítómintán épített fa legjobb részfája mennyire illeszkedik a teszthalmazra.

Ha egy tényleges osztályozási feladatot kell megoldani, ahol az egyik adathalmaz osztályváltozója nem ismert (vagy ismert, de nem optimalizáltak erre a halmazra, hanem csupán a modell eredményességének mérésére szolgál), akkor a tanítóhalmazt két részre kell bontani, ezek alkotják majd a tanító- és a teszthalmazt és az ismeretlen osztályváltozójú halmaz a validáló halmaz.

6.2.2. Eredmények

A 6.1.2. pontban leírt beállítások mellett alkalmaztam az utónyesést. Teszteltem egyrészt a benchmark adatokon, ahol az előző pontban leírt két alkalmazás közül az előbbivel foglalkoztam, mivel az optimalizált paraméterű más algoritmusok paramétereit is úgy optimalizálták, hogy a teszthalmazt többször osztályozták különböző beállítások mellett, azaz lényegében arra optimalizálták a paramétereket.

Másrészt megnéztem az eredményeket a Ford adatsorokon, ahol a tényleges osztályozási feladatnak megfelelően külön tanító-, külön tesztelő- és külön validáló halmazt használtam.

6.2.2.1. Eredmények a benchmark adatokon

A 6.3. táblázatban látható az algoritmus pontossága nyeséssel és nyesés nélkül. Az eredményeket az egyszeres és a többszörös modellezés esetén is ábrázoltam. A harmadik oszlop mindkét esetben azt mutatja meg, hogy az utónyesés alkalmazásával mekkora javulás érhető el a pontosságban. Az adott beállítások és operátorok mellett ez az a maximális pontosság, ami adatsoronként elérhető ezzel a módszerrel. Mint látható, ez az optimalizációs módszer sokkal kevesebbet javít az eredményeken, mint a többszörös modellezés, igaz a futási ideje is jóval rövidebb. Azokon az adatsorokon javítja fel legjobban az eredményeket, amelyekhez az algoritmus „terebélyes" fát épít, azaz olyan fát, amiben szinte minden csomópontnak van két gyermeke és levelek általában csak az alsó néhány szinten fordulnak elő. Az ilyen fák általában a kevés adat miatti túltanulás eredményei (pl. 50Words), ezért néhány ág lenyesése jelentős javulást eredményezhet.

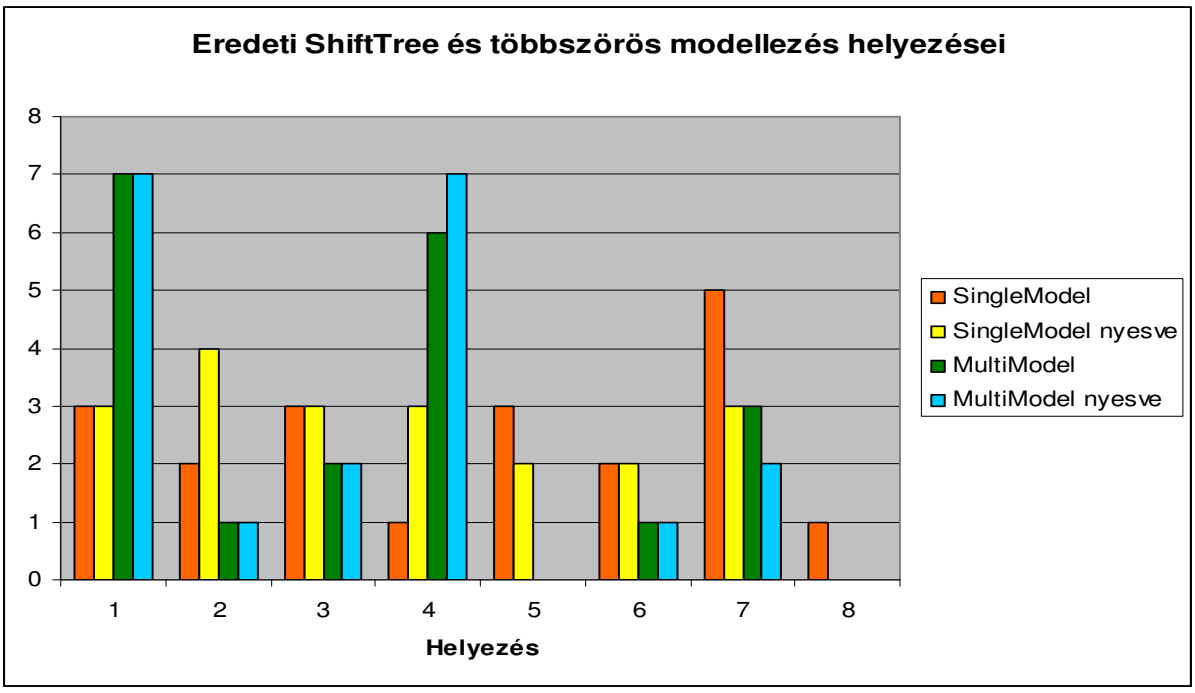
	EREDETI SHIFTTREE			TÖBBSZÖRÖS MODELLEZÉS		
	Nyesés nélkül	Nyeséssel	Javulás	Nyesés nélkül	Nyeséssel	Javulás
50Words	39,78%	46,37%	6,59%	43,96%	49,45%	5,49%
Adiac	52,94%	57,03%	4,09%	52,94%	57,80%	4,86%
Beef	53,33%	60,00%	6,67%	63,33%	66,67%	3,33%
CBF	89,22%	89,22%	0,00%	94,67%	94,67%	0,00%
Coffee	67,86%	75,00%	7,14%	85,71%	85,71%	0,00%
ECG200	82,00%	86,00%	4,00%	88,00%	88,00%	0,00%
FaceAll	61,24%	64,14%	2,90%	70,24%	71,78%	1,54%
FaceFour	71,59%	71,59%	0,00%	84,09%	84,09%	0,00%
Fish	66,29%	70,29%	4,00%	69,71%	72,57%	2,86%
GunPoint	79,33%	80,00%	0,67%	86,00%	86,00%	0,00%
Lighting2	77,05%	77,05%	0,00%	83,61%	83,61%	0,00%
Lighting7	63,01%	67,12%	4,11%	72,60%	72,60%	0,00%
OliveOil	80,00%	80,00%	0,00%	83,33%	83,33%	0,00%
OSULeaf	56,20%	58,26%	2,07%	58,26%	61,57%	3,31%
SwedishLeaf	71,20%	72,96%	1,76%	74,56%	75,84%	1,28%
SyntheticControl	93,67%	93,67%	0,00%	96,00%	96,00%	0,00%
Trace	100,00%	100,00%	0,00%	100,00%	100,00%	0,00%
TwoPatterns	94,18%	94,33%	0,15%	94,93%	94,93%	0,00%
Wafer	98,25%	98,33%	0,08%	98,72%	98,86%	0,15%
Yoga	74,37%	77,07%	2,70%	74,93%	77,80%	2,87%
Átlagosan	*73,58%*	*75,92%*	*2,35%*	*78,78%*	*80,06%*	*1,28%*
Súlyozva átlagosan	*84,80%*	*85,99%*	*1,19%*	*86,85%*	*87,86%*	*1,01%*

6.3. táblázat: A nyesés pontosságnövelő hatása

A 6.4. táblázat mutatja az összesített helyezést az átlagolás és a súlyozott átlagolás alapján. Az egyszeres modellezés esetén nem látszik javulás, bár a 6.3. ábrán megfigyelhető. Ez azzal magyarázható, hogy az átlagosan negyedik és a hatodik helyen álló algoritmusok között viszonylag nagy távolság van pontosság tekintetében, és ugyanez igaz a súlyozott átlag tekintetében második és negyedik algoritmusra.

	HELYEZÉS			
	SingleModel	SingleModel nyesve	MultiModel	MultiModel nyesve
Átlagosan	*5*	*5*	*2*	2
Súlyozva átlagosan	*3*	*3*	*2*	1

6.4. táblázat: Egyszeres és többszörös modellezés (nyeséssel) átlagos pontosságának helye



6.3. ábra: Többszörös és egyszeres modellezés +nyesés összehasonlítása más algoritmusokkal

A 6.3. ábra mutatja az elért helyezések számát. Megerősíti a 6.3. táblázat eredményeit azzal, hogy nincs akkora javulás a nyesés bevezetésével, mint a többszörös modellezéssel. Ugyanakkor az is megfigyelhető, hogy ez a kis javulás is jobbra tolja el a hisztogramot.

Összességében az látszik, hogy az utónyesést a többszörös modellel kombinálva, a súlyozott átlag szerint ténylegesen az élre került a ShiftTree. Az egyszerű átlag esetén is 0,56% hiányzik a győzelemhez, annak ellenére, hogy itt a tanulás-intenzív viselkedés – a kis adatsorok jelentős érvényesülése miatt – halmozott hátrányként jelentkezik.

6.2.2.2. Eredmények a Ford adatsorokon

A Ford adatsorok esetén is a korábban használt konfigurációt használtam egy CBOAVG(5) operátorral kiegészítve. Mivel a fentebb bemutatott módszereket akartam összehasonlítani, nem pedig a legjobb lehetséges eredményt kerestem, ezért itt adatsoronként egy decidert használtam. A FordA esetén ez a DDivFunc(-1/3), a FordB esetében pedig a DDivFunc(1/2).

	FORD(A)			FORD(B)		
	Teszt	**Validáló**	**Helyezés**	**Teszt**	**Validáló**	**Helyezés**
SingleModel	78,48%	81,06%	17	68,18%	64,81%	14
SingleModel nyeséssel	83,03%	82,42%	16	74,55%	69,88%	10
MultiModel	79,70%	80,83%	17	69,70%	65,68%	14
MultiModel nyeséssel	84,24%	82,58%	16	75,15%	70,12%	10

6.5. táblázat: FordA/B eredmények nyeséssel és többszörös modellezéssel

Az eredmények a 6.4. táblázatban láthatóak adatsoronként egyszeres és többszörös modell esetén. Az első oszlop mutatja a teszthalmazon elért legnagyobb pontosságot, a második az ehhez tartozó modellen a validáló adathalmaz eredményét. A harmadik oszlop a Ford Classification Challenge versenyen elért helyezést mutatja, ha a validációs halmazt az adott modellel osztályozva küldtem volna be. Látható, hogy a teszthalmazon mindig nagyobb a javulás, mint a validáló halmazon, de ez nem meglepő, hiszen az előbbire optimalizálunk. Ugyanakkor az is megfigyelhető, hogy a nyeséssel a validáló halmazon is nő a pontosság. Az is észrevehető, hogy mivel a Ford adatsorok tanítóhalmaza nagy, a többszörös modellezés hatása csekély.

7. Fejlesztési lehetőségek (kitekintés)

Ebben a fejezetben azokat a fejlesztési lehetőségeket tekintem át, amelyek az algoritmushoz lazán kapcsolódnak, ugyanakkor vagy a hatékony gyakorlati alkalmazáshoz, vagy az algoritmussal más típusú feladatok megoldásához szükségesek lehetnek.

A fejlesztések első fele a gyakorlatban történő alkalmazáshoz kapcsolódik. A gyakorlati alkalmazások során valószínűleg nem elégséges a 2.5. pontban definiált néhány egyszerű operátor. Érdemes lehet újakat definiálni. Ezek között lehetnének a mostaniakhoz hasonlóak, vagy jelentősen eltérőek. Például olyanok, amik az idősor két megadott változójának a különbségét adja vissza, mint CBO, vagy deriváltat számol, esetleg meghatározza, hogy egy adott szakasz mennyire tér el egy megadott függvénytől; ESO-ként egy adott tulajdonságú szakasz elejére/végére/közepére ugrik, vagy oda, ahol két megadott változó között legnagyobb a különbség; a lehetséges operátorok száma végtelen, csak az alkalmazási területen és fejlesztő fantáziáján múlik.

A többszörös modellezés kapcsán érdemes lehet megvizsgálni, hogy a FIFO sor csomópontjai közül melyik mennyire jelentős, és ennek alapján átalakítani a sort egy prioritásos FIFO sorrá, ahol a később bekerült, de jelentősebb csomópontok megelőzhetik a korábban bekerült, de jelentéktelenebb csomópontokat. Ez egy adatbányászat az adatbányászat felett (datamining over datamining) típusú feladat lenne, ahol az adatsor tulajdonságaitól, a jóságértéktől, a fa egyéb paramétereitől, és a csomópont tulajdonságaitól (pontok száma, osztályok eloszlása, stb.) függően egy nagyon gyorsan működő tanuló eljárásnak meg kéne mondania, hogy mennyire fontos, hogy ezt a csomópontot kifejtsük. Ezen fontosság alapján állapítanánk meg, hogy a csomópont hova kerüljön a sorban, valamint azt, hogy egyáltalán engedélyezzük-e a többszörös modellt az adott csomópontban. Ezzel garantálnánk, hogy a fontosabb ágakat kevésbé akadályozza a modellek számának limite, valamint azt, hogy a futási idő minden beállítás mellett elfogadható legyen.

A fejlesztések másik fele új típusú feladatok megoldására lenne. Az egyik legegyszerűbb fejlesztés az lehet, hogy minden csomópont képes legyen idősor jellegű és hagyományos attribútumok vizsgálatára is. Ezzel lényegében kereszteznénk a hagyományos döntési fát a ShiftTree-vel. Így egyrészt pontosabban tudnánk osztályozni az idősorokat is, hiszen néhány jellemző statisztikai adatot attribútumként az idősor mellé véve több információval rendelkeznénk. Másrészt a módszer alkalmassá válna hibrid adatok osztályozására is, olyan adatrekordokéra, amik a hagyományos attribútumok mellett idősor(oka)t is tartalmaznak.

Az algoritmus alapötlete kiterjeszthető több időtengely irányába is. Ehhez természetesen teljesen új operátorok és adatábrázolás szükséges, de a három részegység és azok feladatai nem változnak. Ezzel a kiegészítéssel a módszert át lehetne alakítani egy képfelismerő rendszerré is. Ezzel a módszerrel olyan modellek lennének építhetőek, amik képesek lennének eldönteni, hogy a képen látható-e valami (pl. a biztonsági kamerák képén egy ember). A módszer természetesen nem korlátozódik két dimenzióra, a kétdimenziós képhez harmadik tengelyként adhatjuk az időt, vagy egy másik koordinátatengelyt. Így képesek lennénk mozgóképek vagy háromdimenziós felvételek (vagy változó 3D felvételek) osztályozására.

Egy másik feladat lehet a mostani algoritmus átalakítása stream jellegű adatok elemzésére, azaz olyan idősorok elemzésére, amik „végtelen” hosszúak. Itt nem a klasszikus osztályozási feladat lenne a jellemző alapvetően, hanem az, hogy meg tudjunk jósolni bizonyos jelenségeket a stream-ben, és az ettől eltérő viselkedésre riasztani, vagy egy előjelből megjósolni valami közeledő hibát, és erre figyelmeztetni a felhasználót.

8. Összefoglalás

Munkám során egy új adatbányászati algoritmust dolgoztam ki. A módszer a klasszikus bináris döntési fákat használja alapként, az egyes csomópontokban három egyszerű részegységgel három egyszerű lépést hajt végre. Ez a három részegység a szemtologató, a feltételállító és a döntő, amiknek a működését a 2. fejezetben láthattuk.

Az algoritmus kidolgozása során három szempontot tartottam igazán fontosnak: az automatizmust, a megmagyarázhatóságot és a pontosságot. Ezek mellett figyeltem a bővíthetőségre és a futási időre.

Az első kritérium, az automatizmus olyan szempontból teljesül, hogy a módszer az egy- és többváltozós idősorokat is képes osztályozni, valamint nem okoznak neki gondot sem az eltérő hosszú idősorok (akár egy osztályozási feladaton belül sem), sem az osztályok száma. Az operátorok definiálása felől nézve az automatizmus sérül, hiszen az algoritmus használójának kell döntenie a definiált operátorokról, azok sorrendjéről és a jóságértékről. Ezek egy részét könnyen implementálható módszerekkel ki lehet küszöbölni. Összességében ez azt jelenti, hogy az adatelőkészítési fázis hossza jelentősen csökkent, viszont a modellezési fázis hossza nem csökkent, a felhasználónak a legjobb kiválasztásához több modellt ki kell próbálnia, mint a legtöbb algoritmus esetében. Ez az ára annak, hogy a módszer egy általános osztályozó, és nem egy speciális problémára lett kifejlesztve.

A megmagyarázhatóság a döntési fa jellegből adódik – és mint a 4.3. alfejezetben láthattuk – a módosított csomópontszerkezet ellenére semmit sem vesztett az erejéből.

A pontossággal kapcsolatban a tapasztalatok azt mutatják, hogy a kevés, egyszerű operátorral rendelkező alapkészlet mellett a ShiftTree valahol a középmezőnyben helyezkedik el, ahogy ezt a 4.2. alfejezetben láthattuk. Az adott problémára specializált algoritmusok általában megelőzik, az általános, paraméteroptimalizálás nélküli osztályozókat viszont általában megelőzi. A többszörös modellezés és az utónyesés bevezetésével az eredmények jelentősen javulnak, az algoritmus az élmezőnybe kerül (6. fejezet). Ráadásul ezek a folyamatok is automatikusak, tehát az elsődleges alapelvet sem sértettem meg a bevezetésükkel. A feladathoz jobban illeszkedő operátorokkal még jobb eredmények érhetőek el. A pontossággal kapcsolatos másik észrevétel, hogy az algoritmus tanulás-intenzív, mint a döntési fák általában, azaz osztályonként sok tanítómintára van szüksége pontos modellek felállításához.

A bővíthetőség – bár csak másodlagos szempont volt – teljes mértékben megvalósult: az egyes modulok kialakítása olyan, hogy bármikor hozzáadhatunk új operátorokat, amik a probléma jobb megoldását elősegíthetik.

A futási idővel kapcsolatban elmondható, hogy gyakorlati mérések alapján egy adott beállítás mellett az idősorok számától négyzetesen függ, illetve a változók számának növekedése maximum négyzetesen növeli a futási időt. A tapasztalatok szerint ez megfelelően gyors.

A módszer továbbfejlesztésére több irány is lehetséges: egyrészt az implementáció bővíthető lenne új operátorokkal, automatizmusokkal. Másrészt a módszert magát át lehetne dolgozni úgy, hogy többdimenziós idősorokat is képes legyen osztályozni, vagy stream-eket elemezni. Ezen kívül egy lehetséges irány olyan hibrid csomópontok kialakítása, amik lehetővé teszik a vegyes (idősor és nem-idősor mezőkkel is rendelkező) adatrekordok osztályozását. (7. fejezet)

Összességében úgy gondolom, hogy az algoritmus legfőbb előnye, hogy a legkülönfélébb idősor-osztályozási problémák megoldására alkalmas, és testre szabható, mint ahogy azt a 4. és 5. fejezetekben láthattuk.

Függelék I. A felhasznált irodalom jegyzéke

[1] **Eamonn Keogh**: A Tutorial on Indexing and Mining Time Series Data. *In ICDM '01: The 2001 IEEE International Conference on Data Mining* (Prezentáció). San Jose, USA, 2001.

[2] **Dr. Abonyi János** (szerk.): Adatbányászat a hatékonyság eszköze. Budapest, 2006, Computerbooks. ISBN 9636183422.

[3] **Dr. Bodon Ferenc**: Adatbányászati algoritmusok. (*tanulmány*) URL: http://www.cs.bme.hu/~bodon/magyar/adatbanyaszat/tanulmany/index.html (2008).

[4] **Nello Cristianini – John Shawe-Taylor**: An Introduction to Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000. ISBN 0521780195.

[5] **Breiman, Leo**: Random Forests. *In Machine Learning 45 (1), 5-32 (2001).*

[6] **Stuart J. Russel – Peter Norvig**: Mesterséges intelligencia modern megközelítésben. Budapest, 2000, Panem Könyvkiadó. ISBN 9635452411.

[7] **Horváth Gábor** (szerk.): Neurális hálózatok és műszaki alkalmazásaik. Budapest, 1998, Műegyetemi Kiadó. ISBN 9634205771.

[8] **Niels Landwehr - Mark Hall - Eibe Frank**: Logistic Model Trees.

[9] **J. R. Quinlan**: Induction of decision trees. *Machine Learning, 1. évf. 1. sz.* ISSN 0885 -6125

[10] **Leo Breiman – Jerome Friedman – Charles J. Stone – R. A. Olshen**: Classification and Regression Trees. 1984. January, Chapman & Hall/CRC. ISBN 0412048418.

[11] **Y.-S. Shin**: Families of splitting criteria for classification trees. *Statistics and Computing, 9. évf. (1999) 4. sz.* ISSN 0960-3174.

[12] **Keogh, E. – Xi, X. – Wei, L. – Ratanamahatana, C. A.**: The UCR Time Series Classification/Clustering Homepage. URL: www.cs.ucr.edu/~eamonn/time_series_data/ (2008).

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

[14] **The University of Waikato**: Weka3 – Data Mining Software in Java. URL: http://www.cs.waikato.ac.nz/ml/weka/ (2008).

[15] **Tony Bagnall**: Weka on Timeseries.
URL: http://www.cs.ucr.edu/~eamonn/time_series_data/WekaOnTimeSeries.xls (2008).

[16] **Adatbányászati Csoport**: DFTMTimeSeries Futási eredmények.
URL: http://adatbanyaszat.tmit.bme.hu/twiki/bin/view/DMTFTimeSeries/WebHome (2008).

[17] **Mahmoud Abou-Nasr – Lee Feldkamp**: Ford Classification Challenge Results. URL: http://home.comcast.net/~nn_classification/datasets/Ford Classification Challenge Results_2008.ppt (2008).

[18] **Saito, N.**: Local feature extraction and its application using a library of bases. PhD thesis, Yale University (1994).

[19] **Roverso, D**.: Multivariate temporal classification by windowed wavelet decomposition and recurrent neural networks. *In 3rd ANS International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface*, 2000.

[20] **Chotirat Ann Ratanamahatana – Eamonn Keogh**: Making Time-series Classification More Accurate Using Learned Constraints.

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

[22] **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.

[23] **M. Kudo – J. Toyama – M. Shimbo**: Multidimensional Curve Classification Using Passing-Through Regions. *Pattern Recognition Letters, Vol. 20, No. 11--13, pages 1103--1111*. (1999).

Függelék II. A módszer kapcsán bevezetett új fogalmak

Fogalom	Magyarázat
CB	A **C**ondition**B**uilder rövidítése
CBE	A **C**ondition**B**uilder **E**xtension rövidítése
CBO	A **C**ondition**B**uilder **O**perator rövidítése
Conditionbuilder	A feltételállító másik neve
Conditionbuilder extension	A feltételállító segéd másik neve
Conditionbuilder operator	A feltételállító operátor másik neve
Decider	A döntő másik neve
Döntő	A ShiftTree csomópontjainak harmadik és egyben utolsó rétege (modulja). Feladata a modellezés során kiválasztani a lehetséges vágási helyek közül a legjobbat. Ehhez a benne definiált jóságértéket használja. Emellett feladata a legjobb vágáshoz tartozó feltétel kiszámítása. Az osztályozásban nem vesz részt. Működésének pontos leírása a 2.1.4. a 2.2. és a 2.3. fejezetekben található. Az implementált verzióban található típusok leírása a 2.5.3. fejezetben található.
ES	Az **E**ye**S**hifter rövidítése
ESO	Az **E**ye**S**hifter **O**perator rövidítése
Eyeshifter	A szemtologató másik neve
Eyeshifter operator	A szemtologató operátor másik neve
Feltételállító	A ShiftTree csomópontjainak második rétege. Feladata idősorokhoz a szem által mutatott érték alapján az operátoraival különböző értékeket számítani. Ezek alapján a modellépítés során szintén feladata a vágási helyek definiálása. Az osztályozás során a kiválasztott operátorával számított értéket hasonlítjuk össze a feltételértékkel. Működésének pontos leírása a 2.1.3. a 2.2. és a 2.3. fejezetekben található.
Feltételállító-operátor	A feltételállító által tartalmazott operátorok összefoglaló elnevezése. Az implementált verzióban található operátorok leírása a 2.5.2. fejezetben található.
ShiftTree	Az idősor-osztályozó módszer elnevezése. Nevét a szemnek nevezett pointer mozgatásáról és az alapját alkotó döntési fáról kapta. Az algoritmus által épített modelleket is hívhatjuk ShiftTree-nek.
Szem	Az idősor változóira egy adott időpillanatban mutató pointer. Az idősor egy adott elemére mutató pointer
Szemtologató	A ShiftTree csomópontjainak első rétege. Feladata a szem beállítása az idősorokon különböző operátoraival. Működésének pontos leírása a 2.1.2. a 2.2. és a 2.3. fejezetekben található.
Szemtologató-operátor	A szemtologató által tartalmazott operátorok összefoglaló neve. Az implementált verzióban található operátorok leírása a 2.5.1. fejezetben található.

Függelék III. Ábrák jegyzéke

Függelék IV. Táblázatok jegyzéke