



Nyugat-magyarországi Egyetem
Simonyi Károly Műszaki, Faanyagtudományi és Művészeti
Kar
Informatikai és Gazdasági Intézet

Adatelemzési rendszer fejlesztése erdészeti elemzésekhez RapidMiner és R környezetben

Kiss Bálint

TDK dolgozat

Témavezetők:

Dr. Jereb László, SKK Informatikai és Gazdasági Intézet
Pödör Zoltán, EMK Matematikai Intézet

2014. április 10.



Nyugat-magyarországi Egyetem
Simonyi Károly Műszaki, Faanyagtudományi és
Művészeti Kar
H-9401 Sopron, Bajcsy-Zs. u. 4. Pf.: 132.
Tel: +36 (99) 518-101 Fax: +36 (99) 518-259

NYILATKOZAT

Alulírott **Kiss Bálint** (neptun kód: **BL0HDS**) jelen nyilatkozat aláírásával kijelentem, hogy az **Adatelemzési rendszer fejlesztése erdészeti elemzésekhez RapidMiner és R környezetben** című

TDK dolgozat

(a továbbiakban: dolgozat) **önálló munkám**, a dolgozat készítése során betartottam a szerzői jogról szóló 1999. évi LXXVI. tv. szabályait, különösen a hivatkozások és idézések tekintetében.

Hivatkozások és idézések szabályai:

Az 1999. évi LXXVI. tv. a szerzői jogról 34. § (1) és 36. § (1) első két mondata.)

Kijelentem továbbá, hogy a dolgozat készítése során az önálló munka kitétel tekintetében a konzulenszt illetve a feladatot kiadó oktatót **nem tévesztettem meg**.

Jelen nyilatkozat aláírásával tudomásul veszem, hogy amennyiben bizonyítható, hogy a dolgozatot nem magam készítettem, vagy a dolgozattal kapcsolatban szerzői jogsértés ténye merül fel, a Nyugat-magyarországi Egyetem megtagadja a dolgozat befogadását és ellenem fegyelmi eljárást indíthat.

A dolgozat befogadásának megtagadása és a fegyelmi eljárás indítása nem érinti a szerzői jogsértés miatti egyéb (polgári jogi, szabálysértési jogi, büntetőjogi) jogkövetkezményeket.

Sopron, 2014. április 10.

.....
hallgató

Tartalomjegyzék

1.	Bevezetés.....	1
2.	Elemzési célok és feltételek	1
2.1.	Fejlesztési célok, motivációk	1
2.2.	Cyclic Reverse Moving Intervals Techniques – CReMIT módszer	2
2.3.	Fejlesztési környezet és felhasznált eszközök.....	3
2.4.	Fejlesztési feladat	4
3.	Egységes elemzési folyamat.....	5
3.1.	Folyamat architektúra.....	5
3.2.	Adatelőkészítő modul.....	6
3.3.	CReMIT modul	6
3.4.	Outlier modul	7
3.5.	Elemző modulok	9
4.	Elemzési folyamat automatizálása	12
4.1.	Elemzések automatizációja RapidMinerben	12
4.2.	Korrelációs sorok összefűzése és szignifikáns r -értékek	14
5.	CREMIT Extension fejlesztése	18
5.1.	RapidMiner adatszerkezetek és objektumai.....	18
5.2.	Kezelőfelület	19
5.3.	Operátor osztály	23
5.4.	CReMIT módszert alkalmazó rekurzív algoritmus.....	28
6.	Elemzési esettanulmányok	33
6.1.	Rovarfogási adatok elemzése	33
6.2.	Levélvesztési adatok elemzése.....	36
7.	Tapasztalatok, továbblépési lehetőségek.....	38
	Köszönetnyilvánítás	39
	Irodalomjegyzék.....	40

1. Bevezetés

A dolgozatban bemutatásra kerülő tevékenységem idősorok elemzéséhez köthető módszer kidolgozásához és fejlesztéséhez kapcsolódik. Periodikus idősorok vizsgálatához létrehozott transzformációs-, és adatelemzési módszer kapcsán végeztem fejlesztéseket, majd ezeket erdészeti jellegű adatsorokra alkalmazva elemzéseket hajtottam végre és értékelhető eredményekre jutottam. A fejlesztések során komoly statisztikai ismeretekre, RapidMiner kezelési- és R programozási tapasztalatokra tettem szert. Bővebben megismertem a RapidMiner szoftver belső adatszerkezeit és működését miközben egy új programkomponenst fejlesztettem hozzá Java nyelven a fenti témában.

A dolgozat alapvető célja az elkészült adatelemzési rendszer struktúrájának és módszertani hátterének, valamint a rendszer elkészítéséhez felhasznált szoftveres technológiáknak a bemutatása. Nem volt ugyanakkor cél erdészeti következtetések megfogalmazása, a kapott eredményeket csak a kialakított technika illusztrálására használtam fel.

2. Elemzési célok és feltételek

Az alapfeladat az volt, hogy különböző erdészeti adatok és a klímajellemzők összefüggéseinek feltárásához készítsek egy adatelemző folyamatot RapidMiner felhasználásával, mely magában foglalja az adatok előkészítésétől kezdve, azok megfelelő transzformációin keresztül az elemzési folyamat végrehajtását is.

Az elemzési folyamat általános célú, de jelen esetben a létrehozását definiáló erdészeti alapproblémákhoz köthető különböző rovarfajok éves fogásainak száma és havi időjárási adatok (csapadékmennyiség, középhőmérséklet) közötti összefüggések vizsgálatára készítettem el.

2.1. Fejlesztési célok, motivációk

Az adatelemzési rendszer kifejlesztéséhez a háttér a Föld-rendszer projekt nyújtotta. Feladatom a TÁMOP-4.2.2.C-11/1/KONV-2012-0015 „Föld-rendszer adatok tudományos feldolgozása és az ismeretek társadalmasítása korszerű informatikai eszközök segítségével” című kutatási projekthez, azon belül is a második számú alprojekthez, az „Erdészeti klímahatás vizsgálatok informatikai módszertani hátterének fejlesztése és alkalmazása (2.2)” című résztémához kapcsolódik. A projektbe 2013 júliusában kapcsolódtam be Jereb László és Pödör Zoltán irányításával.

A projekt nyári időtartama alatt egy adatelemzési folyamatot dolgoztunk ki, mely az idősorok közötti kapcsolatok keresését és elemzési lehetőségeit szélesíti ki. A módszer alapját jelentő transzformációs eljárás alapját a CReMIT módszer biztosítja [1]. A módszer fanövekmény és időjárási adatok esetén már több esetben is alkalmazásra került.

Az elemzésben felhasznált eredeti erdészeti adatok rovarfajok éves összesített fénycsapda fogási adatai voltak különböző fajokra és fogási helyekre, míg a meteorológiai adatok OMSZ-től származó havi csapadékösszegek és középhőmérséklet átlagadatokat jelentettek adott koordinátákra. A vizsgálatok célja az egyes fajok populációdinamikájának klímafüggésének vizsgálata volt, azaz az éves fogási adatok és a havi klimatikus jellemzők közti kapcsolatok

keresése. A vizsgálatokat minden fajra és fogási helyre külön-külön elvégeztük, azonban az eredmények esetében törekedtünk általánosabb összefüggések feltárására fajokénti bontásban. Ez motiválta a későbbiekben ismertetésre kerülő fogási helyek korrelációinak összefűzését és együttes vizsgálatát.

Hasonló volt a feladat faegyedek mortalitásának, éves levélvesztési adatainak esetében is, ahol a faegyedek, 13 fafajcsoportba, 6 erdészeti nagytájba és 5 magassági osztályba voltak sorolva. A két feladat között a különbség gyakorlatilag csak a felhasznált függő változóknak volt. Az elemzési folyamat kidolgozása és megvalósítása során törekedtem az univerzalitásra, így a módszer gyakorlatilag bármely periodikus függő, illetve független változók esetében alkalmazható. Az elemző rendszer így nem csak rovarfogások, falevélvesztés és egyéb erdészeti adatokra működik, hanem más területeken is könnyen alkalmazható.

Az elemzés során fontos feltétel volt a felhasznált idősorok periodicitása. Ez biztosítja ugyanis, hogy a módszerünk alkalmazásával képesek legyünk az egyszerű időegységek mellett időben eltolt, késleltetett és különböző hosszúságú időszakok hatásainak képzésére és vizsgálatára. Ilyen időszakok szisztematikus képzését valósítja meg a Cyclic Reverse Moving Intervals Techniques módszer (hivatkozás) a felhasználó által megfelelően definiált paraméterezés mellett.

2.2. Cyclic Reverse Moving Intervals Techniques – CReMIT módszer

2011-ben Edelényi Márton, Pödör Zoltán és Jereb László megállapították, hogy a nyers idősorok, mint függő és független változók közötti összefüggésvizsgálatok sok esetben nem alkalmasak az összes lényeges kapcsolat feltárására.

Nagyon sok vizsgálatban fontos az időben eltolt és különböző szélességű időszakok vizsgálati lehetősége, ezeknek az összefüggésvizsgálatokban való felhasználása. A fenti problémára a szerzők a Cyclic Reverse Moving Intervals Techniques-t, azaz a CReMIT-módszert dolgozták ki. Az eljárás lényege, hogy a független változóra (vagy akár függőre is) a felhasználó által előre definiált paramétereket, egy maximális időbeni visszalépést, egy maximális ablakszélességet, egy kezdőpontot és egy transzformációs függvényt alkalmazva szisztematikus, módon előállítja az összes lehetséges aggregált adatsort.

Az eredeti eljárás alapparaméterei a következők:

- a) **Kezdő_év:** kezdő év definiálása.
- b) **Befejező_év:** befejező év definiálása.
- c) **Kezdő_melyik év:** a kezdő ablakban hány évvel történjen visszalépés az aktuális évhez képest?
- d) **Kezdő_hónap:** Hányadik hónaptól induljon az aggregáció.
- e) **Befejező_melyik év:** A befejező ablakban legyen-e visszalépés vagy az aktuális évben fejeződjön be.
- f) **Befejező_hónap:** Hányadik hónapig tartson az aggregáció.
- g) **Max_ablakméret:** Egy ablakba maximum hány hónapot vonjunk be, mielőtt a következő ablakot hozzuk létre.
- h) **Művelet:** Egy ablakban található hónapok értékein végrehajtandó aggregáló művelet típusa, ami összegzés, átlagolás, maximum és minimum kiválasztás lehet.

Az eljárás végighalad a teljes idősoron és a fenti paramétereknek megfelelő ablaksorozatokat állít elő a hónapokból. Az első ablakban még egyszerűen csak a kezdőhónap szerepel, adott kezdőhónap mellett a sorozatban következő ablakok hossza mindig eggyel növekszik, a kezdőhónap mellett az azt követő hónapokat sorban egymás után bevonva. Minden egyes ablakon belül a szereplő adatok aggregációja kerül végrehajtásra. Ha az eljárás elérte a maximális ablakméretet, azaz az ablaksorozat utolsó elemét adott kezdőhónap mellett, akkor egy új ablaksorozat képzése kezdődik, melynek első szimpla ablakában az előző sorozat kezdőhónapját követő szimpla hónap szerepel. Az iterációs folyamat addig ismétlődik, amíg a befejező hónap paraméterig nem jut.

Ahogy korábban már említettem, sok esetben fontos az időben eltolt hatások vizsgálata. Ez a gyakorlatban azt jelenti, hogy a képzett időszakok lehetnek akár egy évvel korábbiak is, és átnyúlhatnak két periódus határán. A képzett időszakok jelölésére az alábbi formalizmus került bevezetésre: a szimpla ablakokat egy betű-szám páros jelöli, utalva az időbeli visszalépés mértékére és a vizsgált hónapra, például egy évvel ezelőtti január „p1”, mint „Previous January”, aktuális év decembere pedig „a12”, azaz „Actual December”. Több hónapot tartalmazó ablakokat az ablakon belüli első és utolsó hónap intervallumaként kerül jelölésre, például „p9-a2” az egy évvel ezelőtt szeptembertől a mostani évig tartó február időszakot jelöli.

Az alkalmazott aggregációs művelet feladattól függően lehet összegzés (havi csapadékmennyiségek összege), átlagolás (több hónap középhőmérsékletének átlaga), de akár minimum és maximum értékek meghatározása is.

2.3. Fejlesztési környezet és felhasznált eszközök

Az elemzési módszer megvalósításához és elemzések futtatásához két szoftver került felhasználásra:

RapidMiner

A Rapid-I vállalat által Java programozási nyelven elkészített adatelemző és adatbányászati műveleteket végrehajtó szoftver. Legutóbbi 2013 novemberében megjelent 6.0 verziójuk már fizetős kereskedelmi modellt követett és a vállalatot átnevezték Rapid-I-ről RapidMiner-re. A feladat során felhasznált szoftver 5.3.008 számú verziója még ingyenes és nyílt forráskódú maradt.

R

Nyílt statisztikai-matematikai programozási környezet és szkriptnyelv.

RapidMiner számtalan lehetőséget biztosít az adatsorokon való elemzések (például regressziószámítás, korrelációs mátrix létrehozása, asszociációs, klasszifikációs és klaszterező modellek alkalmazása) végrehajtására, kimutatások, diagramok előállítására és fájlba mentésükre. Letölthető bővítményei lehetőséget biztosítanak a funkcióik kiterjesztésére.

Kezelőfelülete „Drag and Drop”-alapú. A munkaablakban a felhasználó által elhelyezett komponensek megfelelően összekapcsolva egy folyamatot definiálnak, mely az adatok betöltésétől kezdve a vizsgálatok végrehajtásán keresztül egészen az eredmények fájlba írásáig különböző műveleteket hajt végre. Ezeket az alapelemeket RapidMiner-ben operátoroknak nevezik. Az operátorok tetszés szerint paraméterezhetők, végrehajtási sorrendjüket pedig az egymással való összekapcsolásukkal határozzuk meg. Egyes operátorok

egy-egy eredménytáblát generálnak kimenetként. Ezt a RapidMiner környezetben „Example Set”-nek, azaz példa- vagy eredményhalmaznak hívják és az elemzések lefutása után az általunk kiválasztott eredményhalmazok kerülnek megjelenítésre.

A választás részben azért esett erre a szoftverre, mert a Föld-rendszer projekt kereteiben is ezzel dolgozunk, az 5.3.008-as verzió ingyenes, miközben funkcionalitásaiban akár a kereskedelmi forgalomban kapható IBM SPSS Modeler-rel is felveheti a versenyt. Tapasztalt adatelemző szakemberek kezében a RapidMiner hatékony eszköz, ám az adott feladathoz annak lehetőségei kötöttebbek és korlátozottabbak voltak. A CReMIT-módszert és a számunkra fontos elemző komponenseket nehéz lett volna megvalósítani a kezelőfelületével, sok operátor esetén pedig az itt létrehozott elemzési rendszer könnyen átláthatatlanná válhatott volna. Szükség volt egy olyan új eszközre, környezetre, amellyel komplexebb részfeladatok valósíthatók meg, ugyanakkor továbbra is biztosított marad a RapidMinerrel való együttműködés.

Az **R** egy programozási környezet és szkriptnyelv alapvetően statisztikai számításokhoz és ábrázolásukhoz került létrehozásra [2]. Míg a RapidMiner 2001-ben indult, az R már 1993 óta létezik egy külön szervezet, az R-Foundation gondozásában. Szintén számos letölthető bővítménnyel lehet kiterjeszteni, melyek alapvetően statisztikai funkciókat és függvényeket tartalmaznak. Fontos tulajdonsága, hogy a programozó számára nagyfokú szabadságot biztosít, mivel a felhasználók is létrehozhatnak függvényeket, saját programokat vagy akár bővítményeket is. Az R hatékony kihasználásához azonban alapvető programozói ismeretekre is szükség van. A letölthető bővítmények nem mindig tökéletesek, mivel sokat közülük átlag R felhasználók készítenek. A legnagyobb hátránya azonban a nagy memóriaigény, ezért is szokták ajánlani a 64 bites verzióját, mely 4 GB-nál nagyobb memóriát tud megcímezni.

A fent említett okok miatt volt szükség a két eszköz összerendelésére és hatékonyságuk, előnyeik kihasználására. Erre megoldást jelentett a RapidMiner egyik ingyenes kiegészítője, az **R Extension**. Ennek segítségével RapidMiner-ben olyan operátorokat definiálhatók a munkafolyamaton belül, melyekbe R-szkriptet lehet írni, és Rapidminer környezetben futtatni.

A CReMIT eljárást már korábban is egy R-szkript operátor tartalmazta, ám később szükség volt arra, hogy valódi RapidMiner operátor legyen paraméterező felülettel. Ezért egy RapidMiner operátort tartalmazó kiegészítőt írtam **Java** programozási nyelven a **RapidMiner API** nyílt függvényeinek és részkomponenseinek felhasználásával **Eclipse** fejlesztőkörnyezetben.

2.4. Fejlesztési feladat

Fő feladatom egy elemzési rendszer, folyamat létrehozása és további fejlesztések végzése volt a megadott szoftverekkel, elsősorban erdészeti adatok elemzéseikhez. A korábbi pontokban megfogalmazott kiindulópontok és feltételek figyelembevételével a fejlesztési feladat a következőket tartalmazza:

1. A folyamat létrehozásához elsősorban RapidMiner-t, mint adatelemző környezetet kellett felhasználnom. Nem minden funkciót lehet csak szimplán RapidMiner operátorok alkalmazásával hatékonyan megoldani, legyen akár szó a CReMIT eljárás megvalósításáról vagy a például többváltozós regressziós vizsgálatok végzéséről, így a funkciók bővítésére R nyelven írt szkripteket és Java nyelven írt programkomponenseket is be kellett építenem a környezetbe.

2. Az elemzési folyamatnak központi eleme az idősorok kiterjesztését biztosító CReMIT eljárás, amely a gyakorlatban egy külön RapidMiner operátornak kell lennie saját logikával és GUI-s paraméterező felülettel. Kialakított elemzési folyamatnak minél könnyebben bővíthetőnek, fejleszthetőnek kell lennie, ezt pedig a modulrendszerű kialakítás biztosítja.
3. Az elemzések jelenleg alkalmazott technikája a regresszióanalízis, de a folyamatnak alkalmasnak kell lennie arra, hogy később további elemzési technikák is megvalósításra kerülhessenek az egész folyamat újradefiniálása nélkül.

A folyamat kialakítását definiáló problémák között szerepelt a bemeneti nyers adatok sokféleségének egységes kezelési lehetősége. Mind rovarfogások, mind faegyedek levélvesztései esetén más előfeldolgozási módszerekre volt szükség. Az elemzések minél kevesebb felhasználói beavatkozás mellett, kényelmesen, gyorsan és átlátható formában álljanak elő, hogy könnyebb legyen az értékes összefüggéseket megtalálni a függő és független változók között. A folyamat kialakítása során fontos szempont volt, hogy nem csak erdészeti, hanem akár más területeken is általánosságban továbbfejleszthető és alkalmazható legyen.

Ezekre a problémákra megoldást készítettem, mely egységes rendszerben kezeli őket. Az elkészült szoftverek, a létrehozott folyamatok alkalmazhatóságát végül egymástól eltérő erdészeti adatok elemzésével mutatom be.

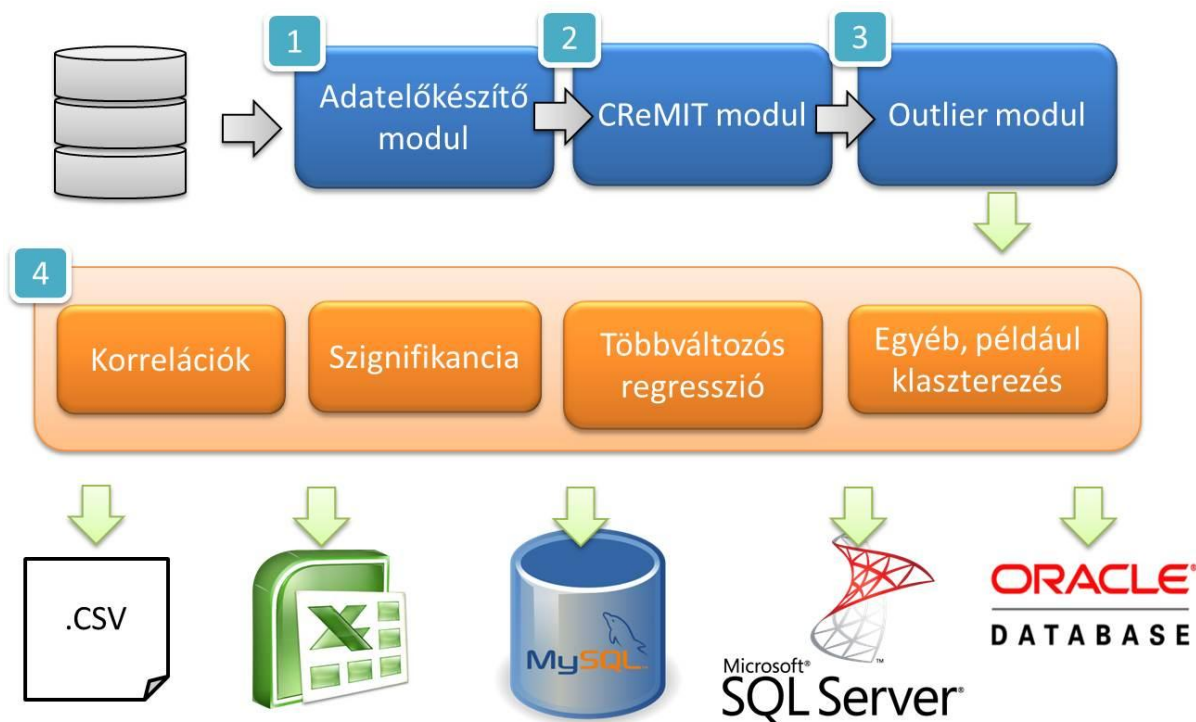
3. Egységes elemzési folyamat

A RapidMiner-ben elkészült elemzési folyamat felhasználja a CReMIT-módszerben alkalmazott aggregálást a független változókon (meteorológiai adatok), és a megfelelő attribútumok közötti lineáris korrelációelemzést végez melyek közül a matematikai értelemben jelentőseket, a statisztikailag szignifikánsakat választja ki és jeleníti meg.

3.1. Folyamat architektúra

A folyamat a nyers adatok betöltése után négy fő részből, modulból tevődik össze: adatelőkészítő modul, CReMIT modul, Outlier-modul és végül az elemző modul különböző lehetőségekkel (1. ábra). A folyamatnak a legnagyobb előnye a rugalmas modulrendszer, a különböző modulok ugyanis cserélhetők, lehetővé téve az egyszerű javításokat, bővítéseket és fejlesztéseket egy modulon belül, miközben a folyamat végére tetszőleges kimenetbe író operátorok helyezhetők el.

A kimenetek táblázatos formában Comma Separated Values (.csv) és Microsoft Excel fájlokba, vagy akár a RapidMiner Write Database operátor segítségével Microsoft SQL, MySQL, PostgreSQL, Oracle, HSQLDB vagy egyéb típusú strukturált adatbázisba írhatók.



1. ábra – A folyamat architektúrája.

3.2. Adatelőkészítő modul

Az adatelőkészítő modul feladata egyrészt a hagyományos értelemben vett adatelőkészítő lépések, mint például adathiányok kezelése, zajok szűrésének végrehajtása. Mivel a CReMIT modul adott szerkezetű adatstruktúrát vár bemenetként, annak kialakítása szintén ennek a modulnak a feladata. Példaként említhető erre a funkcióra, hogy a csapadék adatsorok gyakran eseményjelleggel állnak rendelkezésre, melyből periodikus, például havi adatokat kell képezni. Ugyanakkor a modul más jellegű előkészítő lépéseket is tartalmaz, a rovarfogási adatok előkészítésének esetére az Euklidesz eljárás került kidolgozásra, mely a fogási hely és n darab hozzá legközelebbi koordináta-hoz tartozó meteorológiai mérési adatok alapján végez interpolációt. A faegyedek mortalitásának vizsgálati esetére pedig egy olyan előkészítő lépés került megvalósításra, mely a megmaradt faegyedek darabszáma alapján súlyozott átlagolást végez a levélveszteség mennyiségére és meteorológiai adatokra egy-egy nagytájon, fafajcsoporton és magassági osztályon belül. E funkciók részletesen is bemutatásra kerülnek a dolgozat 6. fejezetében.

3.3. CReMIT modul

A 2.2. fejezetben bemutatott módszer alapján ez a modul létrehozza az időjárési adatok aggregációit.

A CReMIT módszer a meteorológiai paraméterekre, mint független változókra került alkalmazásra mindegyik bemutatásra kerülő feladat kapcsán. Csapadékok esetén összegzést, középhőmérséklet esetén pedig átlagszámítás került alkalmazásra. A feladatok során

konkrétan legfeljebb 6 hónap szélességű ablakokat használtam és előző év januárjáig léptem vissza.

1. táblázat – Csapadékmennyiségek összegzése ablakonként a CReMIT modullal. A meteorológiai adatok az évszámok és a fogási adatok után jelennek meg, a kezdő ablak előző év januárja (1996 eldobásra került), a maximális ablakszélesség 6.

Year/Month	FOGÁSI_HELY1	p1	p1-p2	p1-p3	p1-p4	p1-p5	p1-p6	p2	p2-p3
1997	6	50,314	73,034	91,073	182,375	311,413	396,171	22,720	40,759
1998	25	13,895	17,388	49,618	88,608	169,494	253,885	3,493	35,723
1999	7	13,935	16,907	61,310	99,980	145,262	239,028	2,972	47,375
2000	37	12,280	39,117	55,387	131,222	211,533	292,730	26,836	43,106
2001	6	20,595	31,096	90,008	122,457	156,672	180,154	10,501	69,413
2002	40	21,477	31,409	84,437	121,203	145,661	181,481	9,932	62,959
2003	44	5,324	35,608	62,238	111,094	149,898	192,138	30,284	56,914
2004	96	31,402	36,472	40,825	78,949	120,364	168,169	5,069	9,422
2005	143	29,439	81,585	142,017	192,464	259,721	406,309	52,146	112,578
2006	26	13,657	56,863	74,861	119,683	163,243	221,775	43,206	61,203
2007	4	44,200	61,186	86,728	159,657	230,235	316,565	16,986	42,528
2008	24	40,545	77,319	147,803	148,204	213,701	249,980	36,775	107,259
2009	42	8,639	13,092	70,006	105,723	141,294	307,041	4,453	61,367
2010	26	65,396	110,093	163,676	183,476	241,964	358,312	44,697	98,280

...

a9	a9-a10	a9-a11	a9-a12	a10	a10-a11	a10-a12	a11	a11-a12	a12
37,980	48,420	113,374	159,244	10,440	75,394	121,264	64,954	110,823	45,870
134,263	208,511	259,561	290,541	74,249	125,299	156,278	51,050	82,030	30,980
50,274	72,694	140,060	199,137	22,420	89,786	148,863	67,367	126,444	59,077
53,429	143,827	199,850	251,235	90,398	146,421	197,806	56,023	107,408	51,385
116,214	131,875	161,195	185,051	15,661	44,982	68,837	29,321	53,176	23,855
46,378	134,923	171,887	225,614	88,544	125,509	179,236	36,964	90,692	53,727
33,047	104,065	125,153	151,053	71,018	92,106	118,006	21,088	46,988	25,900
18,270	78,819	106,253	135,421	60,549	87,983	117,151	27,434	56,602	29,168
47,349	50,898	106,046	167,036	3,549	58,697	119,687	55,148	116,138	60,990
17,799	53,336	83,867	92,436	35,538	66,069	74,637	30,531	39,099	8,568
155,107	215,914	248,214	283,442	60,807	93,107	128,335	32,300	67,528	35,228
37,908	61,380	98,363	152,035	23,472	60,455	114,127	36,983	90,655	53,672
42,417	82,880	135,868	193,264	40,463	93,451	150,847	52,988	110,383	57,396
121,251	159,595	225,414	267,200	38,343	104,163	145,948	65,820	107,605	41,786

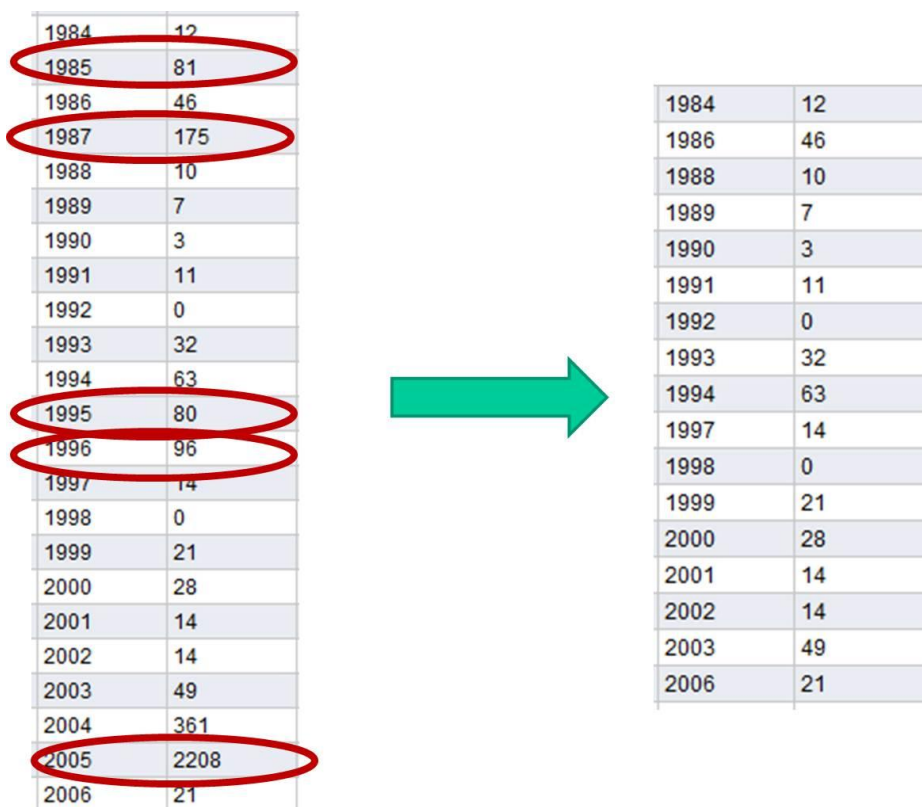
3.4. Outlier modul

Outlier-, azaz kiugró adatnak nevezzük azt az adatot, mely jelentősen eltér, nagy távolságra van a többi adattól. Meghatározásukra sokféle technika létezik, statisztikai alapon, távolság alapján, klaszterezési technikákkal, regressziós technikákkal és még egyéb más módszerrel is. Ilyen kiugró adatot lehet meghatározni a Dixon-féle Q-tesztel, illetve Grubbs-féle tesztel is.

A különböző elemzési eljárások, köztük a regresszióelemzés is érzékeny lehet a kiugró adatokra, torz képet adhatnak az összefüggések. Az Outlier modul célja, hogy az adatsorokban esetleg fellépő kiugró adatok kiszűrésre kerüljenek és az elemzési eljárás elvégezhető legyen mindkét esetre: az elemzéseket lefuttattam kiugró adatok szűrésével és anélkül is. Több módszert is implementáltam, melyek közül a felhasználó választhat.

Bár alapvetően előkészítő technika, és látszólag úgy tűnik a CReMIT modul előtt is tudna szűrni például kiugró rovarfogás adatokat tartalmazó sorokat, viszont a szűrés közben egész sorokat dob el az elemzés egész folyamatából, amire esetleg a CReMIT modulnak szüksége

lehet korábbi évre visszamenő ablakok létrehozásakor. A hiba elkerülése miatt került az outlier szűrés a folyamat azon szakaszába, amikor már előálltak a periodikus adatsorok az előző évi ablakaikkal is.



2. ábra – Évi fogások száma kiugró adatok szűrése előtt és után.

***IQR*-módszer**

Ez egy elterjedtebb és gyakran alkalmazott módszer. Kiszámítjuk az alsó (Q_1) és felső (Q_3) kvartilist, vesszük ezek különbségét, az interkvartilis terjedelmet (IQR), majd azt tekintjük kiugró adatnak, melyek a felső kvartilistól lefele és alsó kvartilistól felfele legalább $1.5 * IQR$ -rel térnek el:

$$Q_1 - 1,5 * IQR < x_i < Q_3 + 1,5 * IQR.$$

Scores-eljárás, Z-próba

Az elemzéseink során felhasznált statisztikai programozási környezet, az R rendelkezik egy „scores”-eljárással[3]. Az eljárás Z-próbát használ (innen az elnevezés, „Z-scores”), mely a mintaátlag és szórás értékek felhasználásával vizsgálja, hogy kiugró adatról van-e szó, vagy sem. Egy megfigyelés Z_i értékét úgy kapjuk meg, hogy vesszük a vizsgált kiugró érték és a minta átlagának a különbségét, majd a minta szórásával elosztjuk. Képlete:

$$Z_i = \frac{x_i - \bar{x}}{S}$$

A kapott Z_i -értéket a próba során összehasonlítja a szignifikancia szint és a mintanagyság alapján számított kritikus értékkel, ennek függvényében dönti el, hogy a vizsgált adat kiugró-e vagy sem.

A Z-próbán alapuló kiugró adatkeresés alkalmazása lehet több kiugró adat egyidejű keresésére is, azonban ennek eredménye félrevezető lehet különösen kis elemszámú minták esetében [4].

Átlag és konstans * szórás különbsége

Hasonló az *IQR*-módszerhez, itt az átlagtól legalább a minta szórásának c konstansszorosával eltérő elemeket tekintjük kiugró adatnak. Képlete:

$$\bar{x} - c * S_x < x_i < \bar{x} + c * S_x.$$

Az átlag és a szórás értéke viszont pont a kiugró adatok miatt esetleg torz lehet, ami félreviheti ezt a módszert.

3.5. Elemző modulok

Az erdészeti jellegű adatok elemzéséhez több lehetséges módszer is rendelkezésre áll, például válaszfüggvény-elemzés, klaszteranalízis és evolúciós-elemzés [5]. Cél az adatsorok közti kapcsolatok vizsgálata volt, például hogy elemezhetjük az egyes nagylepkefajok populációdinamikájának klímafüggését. Az elemző modulban jelen vizsgálatok kapcsán megvalósított eljárás a lineáris korrelációelemzés, azonban egyéb eljárások és módszerek megvalósítása is fejlesztés alatt áll.

Korreláció-és regresszióanalízis

A regressziószámítás egy olyan statisztikai eszköz, mellyel két vagy több változó mennyiség (például y : rovarfogások száma, x : augusztusi középhőmérséklet) közötti kapcsolatot lehet függvényesítve meghatározni. A korrelációs együtthatók pedig ezen kapcsolatok erejének számszerű jellemzésére szolgálnak. Az alkalmazott modell lehet egy vagy többváltozós a vizsgálatba bevont független paraméterek számától függően.

Kétváltozós modellek

A kétváltozós lineáris regresszió segítségével előállítható egy modell, mely szemléletesen azt mutatja, hogy ha a vizsgált adatokat egy koordináta-rendszerben pontokként jelölnénk, akkor a pontokra melyik lineáris egyenes illeszkedik a legjobban például a legkisebb négyzetek elve alapján.

Korrelációanalízis során meghatározott korrelációs együttható egy olyan érték, mely a pontok közti lineáris kapcsolat erejét és irányát mutatja, vagyis a pontok mennyire illeszkednek a regressziós egyenesre. Értéke -1 és 1 között lehet. Minél közelebb van az 1 abszolút értékéhez, a kapcsolat annál erősebb, ellenben nullához való közeledés esetén a két változó kevésbé függ egymástól, nincs köztük (jelentős) lineáris kapcsolat [6]. Az együttható előjele pedig a kapcsolat irányára utal: pozitív előjel esetén egy azonos irányú, negatív előjel esetén pedig egy ellentétes irányú lineáris kapcsolatról beszélhetünk. Az elv kiterjeszthető többváltozós esetekre is, azonban ekkor a korrelációs együttható helyett már az illeszkedés jószágát mérő determinációs együtthatóról szokás beszélni és a szemléletes megjelenítés is nehezekebb.

A kétváltozós lineáris regressziót alkalmazó modul egy-egy sorból álló táblázatokat hoz létre, mely minden előállított CReMIT ablakhoz tartozó előjeles r -értékét tartalmazza.

2. táblázat – Kétváltozós lineáris regresszió kimenete, mely egy sort tartalmaz a település nevével és az ablakok r értékeivel.

Település	p1	p1-p2	p1-p3	p1-p4	p1-p5	p1-p6	p2	p2-p3	
FOGÁSI_HELY1	-0,082	0,140	0,123	0,090	-0,016	0,153	0,322	0,213	
...									
a9	a9-a10	a9-a11	a9-a12	a10	a10-a11	a10-a12	a11	a11-a12	a12
-0,389	-0,347	-0,359	-0,288	-0,065	-0,104	0,016	-0,067	0,086	0,222

Szignifikáns korrelációk kiválasztása

Az adathalmazban függő és független változók közötti szignifikáns korrelációkat keres és táblázatos formában megadja melyik hónaphoz, ablakhoz tartozik. A három szignifikancia szint, melyre lefut a kiválasztás $\alpha = 0,1$ -es $\alpha = 0,05$ -ös $\alpha = 0,01$ -es valamint egy negyedik tetszőleges α szignifikancia szint is definiálható. A szignifikáns r -értékek meghatározása $n-2$ szabadsági fokszámú t -próbával történik, ahol n a vizsgált sorok, évek száma.

3. táblázat – Kiválasztott szignifikáns korrelációk a három szignifikancia szintre.

Hónap/Ablak	alfa = 0.1	alfa = 0.05	alfa = 0.01
a2-a4	0,274		
a3-a4	0,242		
a4	0,435	0,435	0,435
a7-a8	0,334	0,334	
a7-a9	0,280		
a8	0,409	0,409	0,409
a8-a9	0,283	0,283	

Többváltozós regressziós vizsgálatok

Nem mindig elégséges csak kétváltozós vizsgálatokat alkalmazni. Ha csak egyszerűen egy adott y függő és x független változó közötti kapcsolat korrelációs együtthatóját akarjuk kiszámítani, az nem nehéz feladat, ám nem mindig olyan eredményt ad, ami nekünk megfelel. Hiszen a gyakorlatban a függő változókra általában több független változó is komoly hatást gyakorolhat. A függő változóval önmagukban szoros kapcsolatban lévő független változók a többváltozós lineáris regressziós modellbe bevonásán alapuló egyszerű megközelítés nem biztos, hogy az optimális modellt eredményezi.

Ennek egyik lehetséges oka a multikollinearitás [7], azaz a független változók közötti kapcsolat léte. Annak elkerülésére, hogy ilyen változók kerüljenek bevonásra érdemes egy olyan eljárást használni, ami az adott változó hatásának számításakor a többi változót is figyelembe veszi. Erre lehet egy megoldás a parciális korreláció, ami azt mutatja meg, hogy milyen szoros a kapcsolat valamelyik kiválasztott tényezőváltozó és a függő változó között, ha a többi tényezőváltozó hatását mind a vizsgált tényezőváltozóból, mind az eredményváltozóból kiszűrjük. Ezenkívül érdemes csak olyan változókat bevonni, melyek szignifikáns javulást hoznak a modellben.

R-ben léteznek erre beépített függvények, nevezetesen például a StepAIC, mely Akaike-féle információs kritérium alapján hoz létre modellt Stepwise algoritmussal. A Forward Stepwise módszer egy olyan speciális szelektálás a független változók között, ahol kezdetben nem

szerepel egy tényezőváltozó sem a modellben, majd lépésenként vonunk be olyan változókat, mely szignifikánsan javítják a regressziós modellt [8].

Az említett R komponensek korlátját jelentette azonban, hogy nem képes túl sok változót kezelni, így szükséges volt egy saját eljárás létrehozása a Forward Stepwise regressziós eljárás megvalósítására.

Az algoritmus kezdetben a CReMIT módszerrel előállított aggregált csapadék- és hőmérséklet adatokból, mint független változókból megkeresi azt az attribútumot, mely korrelációs együtthatójának négyzete a legnagyobb és bevonja a független változók listájába első elemként. Ezzel egy lehetséges egyváltozós kezdeti lineáris regressziós modellt állít elő, ahol F -próbával vizsgálja, hogy a bevont változó szignifikáns kapcsolatban van-e az eredményváltozóval, majd t -próbával ellenőrzi a paraméter együtthatója 0-tól való eltérésének szignifikanciáját. Amennyiben az F -próba sikertelen, a modellépítésnek vége. Ha sikeres, de a t -próba sikertelen akkor veszi a következő lehetséges változót és addig folytatja, míg vagy talál egy megfelelő kezdő elemet, vagy már itt leáll az eljárás.

Ha a kezdeti modell ellenőrzése sikeres, akkor létrehozta az egy független változót már biztosan tartalmazó kezdő modellt. Ezután végighalad a maradék attribútumokon és következőnek azt veszi bele a felhasználandó független változók listájába, melynek a függő változókkal számított parciális korrelációja a legnagyobb. Új regressziós modellt állít elő és varianciaanalízissel összehasonlítja az előző modellel, eldöntendő, hogy szignifikánsan javult-e a modell és együtthatója is megfelelő-e.

A varianciaanalízis (ANalysis Of VAriance, ANOVA) a két modell össz-szórásnégyzetei alapján vizsgálja a két modellt. Az algoritmus a vizsgálatból veszi az F -próba eredményét, az új modellből pedig az összes bevont független változó t -próbáinak értékeit, és ez alapján dönti el az eljárás, hogy az aktuálisan vizsgált független változó bevonandó-e ténylegesen a modellbe. Ezt ismétli addig, amíg el nem fogynak a bevonandó változók, vagy ha további változók bevonása már nem javítja szignifikánsan a többváltozós modellt.

Ezt a modult is egy R-szkript operátorban valósítottam meg. Kimenete egy táblázat, melyen végig lehet követni a többváltozós modell felépítésének menetét, egyes lépésekbe mely független változók kerültek bevonásra. A kimenetben R^2 oszlop jelöli az adott időben bevont ablakokkal alkotott regressziós modell R^2 értékét, *const* a regressziós egyenlethez megfelelő konstans, az ablakok alatt pedig az ablakok r értékeinek változása szerepel a modellbe bevonáskor (4. táblázat).

4. táblázat – Többváltozós regressziós vizsgálat kimenet.

R2	const	p11.csapadek	p5-p7.csapadek	p5-p7.homerseklet	a5.csapadek	a4.csapadek	a8-a10.homerseklet	a6.homerseklet
0,479	1,445	-0,009						
0,817	1,088	-0,014	0,003					
0,919	-0,960	-0,013	0,003	0,034				
0,988	-1,695	-0,012	0,003	0,043	0,003			
0,993	-1,652	-0,012	0,003	0,043	0,003	-0,001		
0,997	-2,521	-0,012	0,003	0,044	0,004	-0,002	0,017	
0,999	-2,430	-0,012	0,003	0,048	0,005	-0,003	0,021	-0,026
1,000	-2,283	-0,012	0,003	0,046	0,005	-0,003	0,017	-0,024
1,000	-2,232	-0,013	0,003	0,045	0,005	-0,003	0,017	-0,024
1,000	-2,241	-0,013	0,003	0,044	0,005	-0,003	0,016	-0,022

Egyéb megvalósítható elemzések

Bár nincsenek még megvalósítva, további lehetőségek is vannak még elemzések elvégzésére, mint például klaszteranalízis, válaszfüggvény-elemzés és evolúciós módszerek. Klaszteranalízis esetén az adatsorok alkalmas hasonlóság mérték alkalmazásával egymástól elkülönülő csoportokba, klaszterekbe kerülnek. A válaszfüggvény-elemzés egy olyan többváltozós elemzési technika, mely kijelöli a célváltozóra szignifikáns hatással bíró magyarázó változókat és meghatározza a feltárt kapcsolatok erősségét és irányát. A mozgó intervallumos és evolúciós módszerekben kezdetben nem a teljes adatsor kerül vizsgálatra, hanem egy rövidebb hosszúságú adatsort lépésenként növel a vizsgálatok során, vagy lépésenként mozgat előre [9]

4. Elemzési folyamat automatizálása

Viszonylag kis méretű adathalmazok esetében is problémát jelenthet már, hogy ha az elemzéseket sok egymástól némileg eltérő esetben futtatja a felhasználó, mondjuk összes vizsgálandó rovarfajra vagy faegyedek esetén összes nagytájra és magassági osztályra, akkor egy-egy eset után mindig manuálisan újra kell paraméterezni a folyamatot és elindítani a következő elemzést. A hatékonyság és kényelem szempontjából minél inkább automatizáltabban kell működnie az elemzési folyamatnak, hogy minél kevesebb beavatkozást igényeljen a teljes elemzési folyamat lefuttatása.

Erre adnak megoldást a RapidMiner egyes részeinek alfolyamatokba történő beágyazása. Az automatizálás segítségével pedig a végén a lefutott elemzések után egy összefűzött táblázatot állítottam elő, mely tartalmazza a településenkénti korrelációs sorok összefűzését akár a szignifikáns értékeket is kiemelve. Az alábbi eset a rovarfogások elemzésének automatizálását mutatja, de ugyanilyen automatizációs módszert alkalmaztam faegyedek elemzése során is.

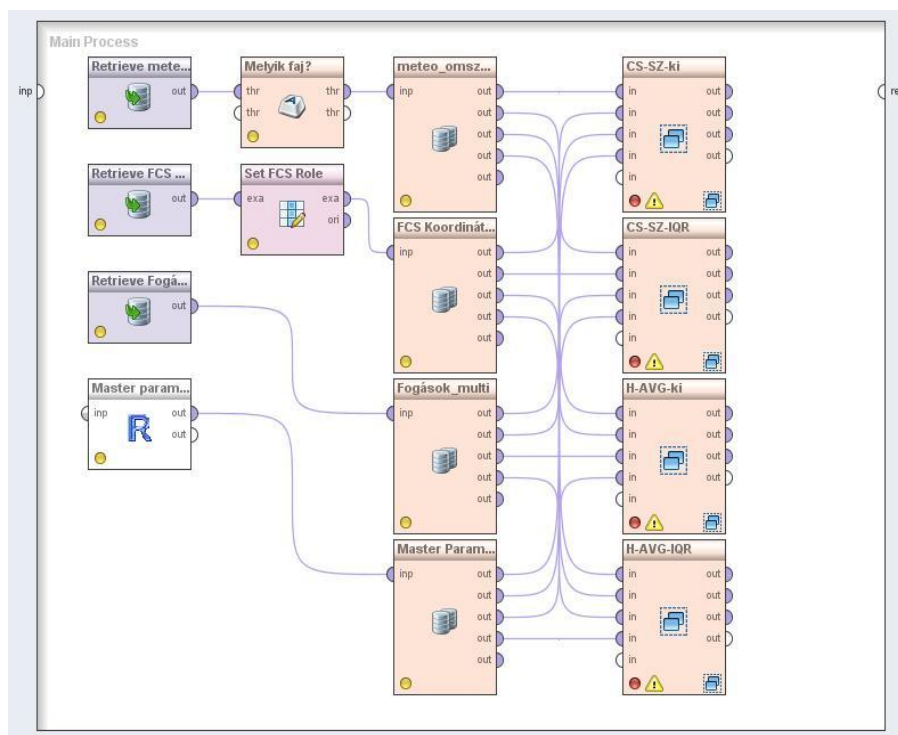
4.1. Elemzések automatizációja RapidMinerben

RapidMiner-ben egy elemzési folyamaton belül definiálhatóak alfolyamatok (subprocess) is. Alfolyamatok a Subprocess-operátorral hozhatók létre, az operátoron belül pedig úgy hozhatók létre műveleteket végrehajtó operátorokat, mint a normál folyamatok esetében.

Első lépésként a következő elemzési folyamatokat automatizáltam:

1. Csapadékmennyiségek CReMIT-es összegzése outlier-szűrés nélkül
2. Csapadékmennyiségek CReMIT-es összegzése outlier-szűréssel
3. Hőmérséklet CReMIT-es átlagolása outlier-szűrés nélkül

4. Hőmérséklet CReMIT-es átlagolása outlier-szűréssel



3. ábra – Meteo adatok, aggregáló műveletek és outlier szűrések automatizálása: Csapadék-Szumma-ki, Csapadék-Szumma-IQR-módszer, Hőmérséklet-Átlag-Ki, Hőmérséklet-Átlag-IQR-módszer

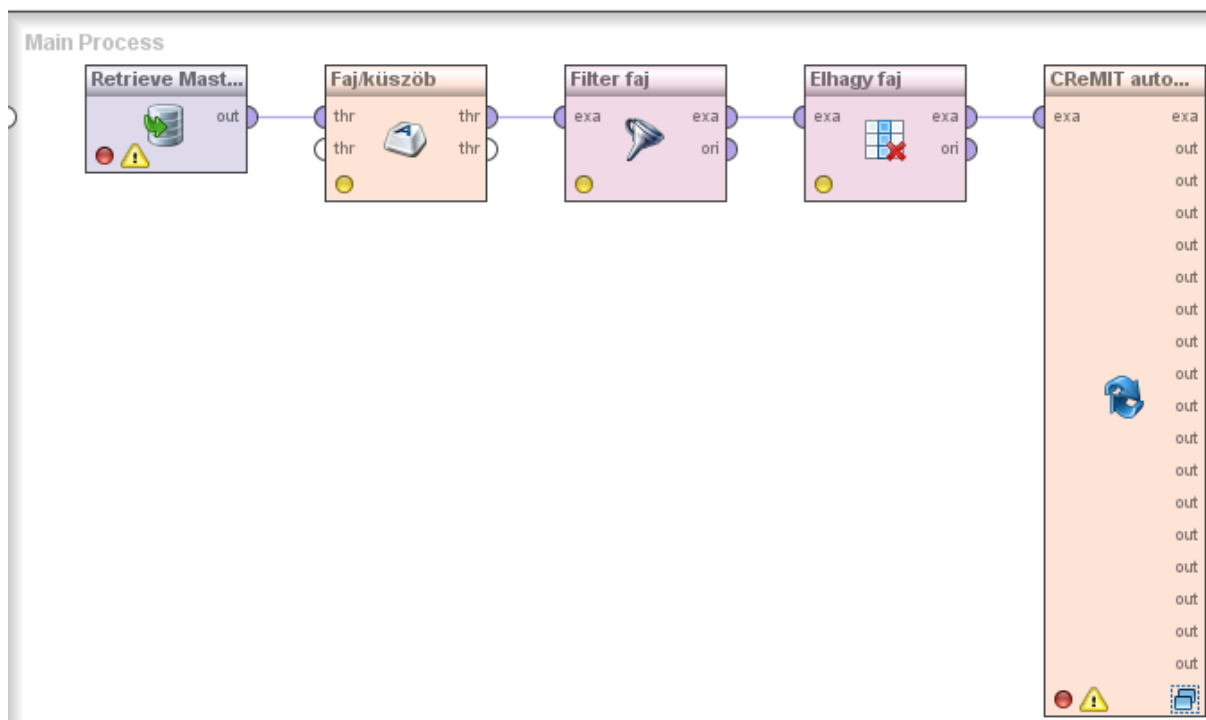
Ezekhez négy alfolyamatot hoztam létre, az alfolyamatokba pedig bemásoltam az eredeti elemzési folyamatot, majd mind a négy paraméterezését átállítottam a fentiek szerint. A nyers adatok betöltése pedig az alfolyamatokon kívülre kerültek, akár csak egy „mesterparaméterezés” R-szkript, melyben a települést és évszámokat kell megadni.

Ez még csak az első lépés, még ennél egy szinttel fentebb is lehet menni a települések és évszámok automatizálásával. Ehhez először egy .csv-fájlba gyűjtöttem a rovarfajokhoz tartozó településeket és évszámokat, így ezek a paraméterek ebből a fájlból töltődnek be. A települések és évszámok betöltése után egy makró-operátoron belül kell megadnia a felhasználónak, mely rovarfajon kívánja az elemzéseket elvégezni. RapidMiner esetén a makrók változóknak felelnek meg, melyekből az operátorok értékeket tudnak kinyerni. Ebben a makró-operátorban definiáltam egy k küszöböt, mellyel azt határozható meg, hogy csak azok az időszakok jelenjenek meg az eredménytáblában amelyekben legalább k szignifikáns érték található. Jelen vizsgálatban a $k=1$ értéket alkalmaztam. Amennyiben egy CReMIT intervallumra egyetlen fogási helyen sincs szignifikáns érték, akkor az adott CReMIT-es intervallum oszlopa – a jobb áttekinthetőség érdekében – kihagyandó a legvégén az összefűzött korrelációk táblázatából.

Egy szűrő (Filter) operátorral kiválasztom a makróként megadott fajnév alapján a településeket és hozzájuk tartozó évszámokat. Az „attribútum kiválasztása” operátorral (Select Attributes) elhagyom a fajnevek attribútumát, hogy ne zavarjon a további folyamatban.

RapidMiner-en belül ciklusok is létrehozhatók. Ezek valójában alfolyamatok, melyek például egy megfelelő iterátor alapján ismétlődnek. Jelen esetben az iterátor nem más, mint a fogási települések és hozzájuk tartozó évszámok. A ciklus az első lépésben definiált automatizációt tartalmazza és minden ismétléskor azt futtatja le a kapott település és évszám sorokkal.

4.2. Korrelációs sorok összefűzése és szignifikáns r -értékek



4. ábra – A települések automatizálása: .csv-ből „mesterparaméterek” betöltése, makróban faj és k küszöb definiálása, vizsgálandó faj paramétereinek kiválasztása, faj nevének attribútumának elhagyása és a ciklus.

Mivel az automatizálással javított módszerben egy futtatás alatt lezajlik minden fogási helyre az elemzés és a korrelációs értékek előállítása, adott a lehetőség, hogy a településenkénti korrelációk egy egységes táblában jelenítsem meg, a táblának pedig különböző változatai legyenek attól függően, hogy az milyen szinten szignifikáns r -értékeket tartalmaz. Ebben az esetben négy tábla áll elő például a csapadékmennyiség elemzésére outlier-szűrés nélkül: szignifikanciavizsgálat nélküli, valamint $\alpha = 0,1$ -es, $\alpha = 0,05$ -ös és $\alpha = 0,01$ -es szignifikanciaszinten szűrt korrelációk.

Ehhez az eredeti elemző modult tartalmazó elemzési folyamatban módosítanom kellett a korrelációk előállításának módját, hogy egy településre négy korrelációs sor álljon elő. Egy R-szkriptet tartalmazó operátort hoztam létre, mely korrelációt és szignifikanciát számít.

```
#####
# KORRELÁCIÓS SOROKAT ELŐÁLLÍTÓ ÉS SZIGNIFIKÁNS R-ÉRTÉK MEGHATÁROZÓ #
# SZKRIPT #
#####

# Szignifikanciaszámító függvény

szigEldont <- function(x, y, alfa)
{
  # R-érték
  r_ertekek <- get("estimate", cor.test(x, y, conf.level =
                                         (1-(alfa/2)) ))

  # Szabadságfok, n-2
  szabadsagfok <- get("parameter", cor.test(x, y, conf.level =
                                             (1-(alfa/2)) ))

  # Kritikus érték, Student-eloszlás
  kritikus_ertekek <- qt((1-(alfa/2)), df=szabadsagfok)

  # Próbatatiztika
  probastatiztika <- get("statistic", cor.test(x, y, conf.level
                                              = (1-(alfa/2)) ))

  # Ha a próbatatiztika a kritikus értéknél nagyobb vagy
  # egyenlő, akkor IGAZ értéket ad vissza
  if( abs(probastatiztika) >= kritikus_ertekek )
  {
    szigSzur <- TRUE
  } else {
    szigSzur <- FALSE
  }
}
```

A szkript elején egy függvényt írtam, mely két attribútumot kap paraméterül (x, y), ezek kapcsolatáról el kell döntenie, hogy adott szignifikanciaszinten szignifikáns-e, valamint a szignifikanciaszint paraméterét is (alfa) meg kell adni. A függvény ugyanaz, mint ami az eredeti elemző folyamat moduljában is található.

Az r-értéket a beépített *cor.test()*-függvénnyel kapom meg. Itt is az x, y paramétert kell átadni, valamint konfidenciaszintet, mely $1 - (\alpha / 2)$ -ként áll elő. A *cor.test()* egy eredménylistát állít elő, és az „estimate”-mező tartalmazza a számított r-értéket. Ezt nyeri ki lebegőpontos változóként a *get()*-függvénnyel. Szabadságfokot, azaz a DF-et szintén *cor.test()*-el meg lehet kapni, ekkor pedig a „parameter” mezőre van szükség. A *qt()*-függvénnyel az R-környezet Student-eloszlás alapján meghatározza a kritikus értéket, csak függvényparaméterül a konfidenciaszintet és a szabadságfokot kell megkapnia. Próbatatiztika meghatározásához a *cor.test()*-függvény „statistic” tagjára van szükség.

A lefutás előtti elágazás csak egy biztonsági óvintézkedés: ha a bemeneti tábla tartalmazza az évszámok oszlopát, melyre nincs szükség, akkor azt az oszlopot a szkript elhagyja.

```
# Felesleges és zavaró attribútum elhagyása
if("Year/Month" %in% colnames(data))
{
  data <- as.data.frame(data[,!(names(data) %in% "Year/Month")])
}
```

A szkript előállít egy teljes korrelációs mátrixot. Az adott település nevét külön eltárolja, hogy később fel tudja használni. A korrelációs mátrixból csak az első sorát tartja meg (1) és elhagyja az első oszlopát (-1).

```
# Alap korrelációs mátrix első sorának előállítása és település
# nevének kinyerése

korrsor_teljes <- as.data.frame(cor(data), stringsAsFactors = FALSE)
telep_df <- as.data.frame(colnames(korrsor_teljes)[1])
colnames(telep_df)[1] <- "Település"
korrsor_teljes <- korrsor_teljes[1,-1]
```

Az előbb előállított sort módosítás nélkül eltárolja három változóban, miközben három tizedesjegyre kerekíti. Eerre a három sorra végzi el a szignifikáns r -értékek szűrését különböző szignifikanciaszinteken. Végigmegy az eredeti korrelációs sor összes mezején és meghívja rájuk a szkript elején definiált függvényt. Amely korreláció nem szignifikáns, annak az értéke helyére egy üres sztring kerül.

```
# Három szignifikanciaszint korrelációs sorainak előállítása
korrsor_01 <- as.matrix(format(round(korrsor_teljes, 3), nsmall = 3))
korrsor_005 <- as.matrix(format(round(
    korrsor_teljes, 3), nsmall = 3))
korrsor_001 <- as.matrix(format(round(
    korrsor_teljes, 3), nsmall = 3))

for( i in 1:ncol(korrsor_teljes) )
{
    if( szigEldont(data[,1], data[,i+1], 0.1) == FALSE )
        korrsor_01[1,i] <- ""
    if( szigEldont(data[,1], data[,i+1], 0.05) == FALSE )
        korrsor_005[1,i] <- ""
    if( szigEldont(data[,1], data[,i+1], 0.01) == FALSE )
        korrsor_001[1,i] <- ""
}
```

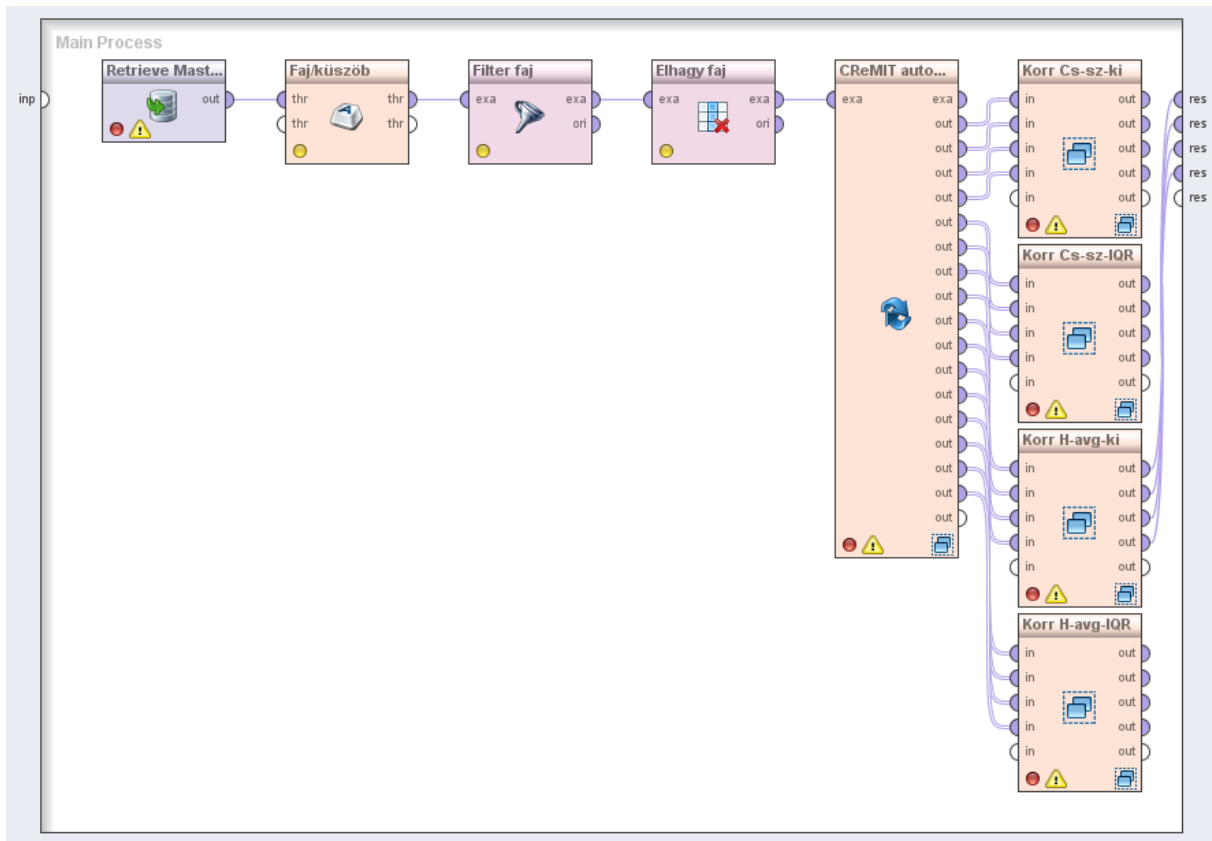
A sorokhoz végül hozzákapcsolja a település nevét, majd átadja a kimenetekbe.

```
# Településnevek hozzáfűzése
result_teljes <- as.data.frame(cbind(telep_df, korrsor_teljes))
result_01 <- as.data.frame(cbind(telep_df, korrsor_01))
result_005 <- as.data.frame(cbind(telep_df, korrsor_005))
result_001 <- as.data.frame(cbind(telep_df, korrsor_001))
```

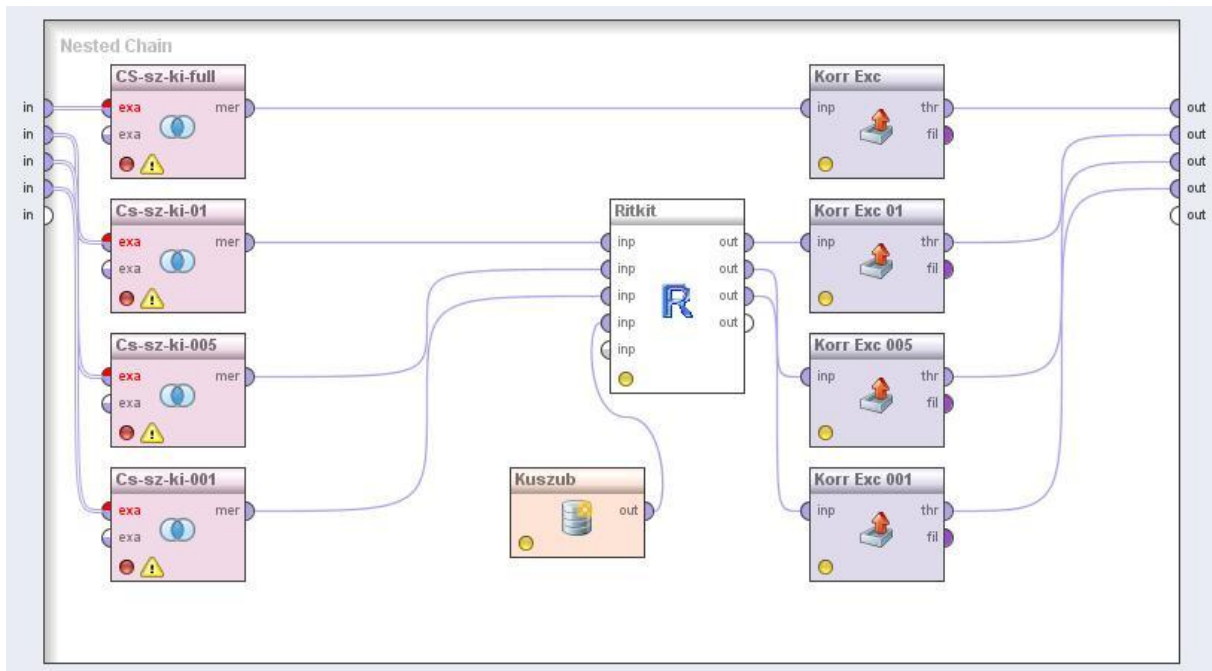
RapidMiner-en belül létezik egy „Összefűzés” (Append) operátor, mely arra szolgál, hogy két bemenetként kapott eredményhalmaz sorait attribútumaik mentén egy eredményhalmazba illesszék össze. Az operátor viszont bemenetként egy ciklus kimeneteit is kaphatja, lépésenként tudja a ciklus eredményhalmazait egy táblába illeszteni.

Létrehoztam a ciklus után négy alfolyamatot a meteorológiai adatok és outlier szűrések miatt. Egy-egy alfolyamatba pedig négy-négy Összefűzés-operátort helyeztem el: a szűrés nélküli korrelációs sorok, valamint az $\alpha = 0,1$ -es, $\alpha = 0,05$ -ös és $\alpha = 0,01$ -es szignifikanciaszinten szűrt korrelációk négy-négy különböző táblákba lesznek összefűzve és fájlokba írva. Az eredmények jobb áttekinthetősége érdekében csak azokat az időszakokat jelenítem meg, ahol a vizsgált fogási helyek legalább egy adott hányadosa szignifikáns kapcsolatot mutat. Ehhez egy R-szkriptet írtam, mely veszi a makróként definiált k küszöböt, és az összefűzött

táblából eldobja azokat az oszlopokat (időszakokat), ahol nincs legalább k szignifikáns r -érték.



5. ábra – Az automatizálás után illesztett négy alflowamat, melyben az összefűzések történnek.



6. ábra – Az összefűzést végző alflowamat tartalma: teljes és szignifikánsan szűrt sorok egy-egy táblába fűzésük, szignifikáns sorok ritkítása és fájlba írás.

5. CREMIT Extension fejlesztése

A CReMIT módszert végrehajtó algoritmus kezdetben R nyelven volt megírva és R-szkript operátorban futott. Korábban nem volt lehetőség például több évre visszalépni vagy hogy ne csak hónapokra működjön, hanem általánosabb időszakokra és attribútumokra is. Legnagyobb hátránya viszont a lineáris és a nehezen áttekinthető forráskód volt, valamint a felhasználóbarát paraméterező felület hiánya: a paramétereket külön táblázatként kapta meg feldolgozásra, amit szintén egy másik R szkript állított elő, amibe bele kellett írni a különböző változónevek mellé az értékeket. Ez redundáns és felesleges lépés, valamint nagyon könnyű hibákat véteni a paraméterek definiálásakor.

Szükség volt egy új operátor definiálására, mely a többi beépített operátorhoz hasonlóan kezelőfelülettel rendelkezik, valamint új struktúrájával kiküszöböli az eredeti R nyelven írt eljárás korlátait és hiányosságait. A RapidMiner nyílt forrású természete és alkalmazás programozási interfészére, API-jára való építkezés mindenkinek lehetőséget ad új kiegészítők és programkomponensek készítésére Java programozással [10]. Így jött létre a CReMIT aggregációt tartalmazó CREMIT Extension kiegészítő is, melyben a Java nyelven írt operátor szerepel. Ezt a kiegészítőt akár publikusan letölthetővé is lehet tenni, ha az Affero General Public License által megadott feltételként a forrása is nyíltan elérhető.

A kész eljárás rekurzív metódushívásokat használ lineáris felépítés helyett, azaz egy metódus önmagának hívásával ismétli működését más átvett változókkal. Az elkészült új operátor továbbá nem csak 12 attribútumig, azaz csak hónapokra képes lefutni, hanem tetszőleges számú attribútumon tud végighaladni, így lehetséges akár napi adatok aggregálása is.

Az új operátort tartalmazó kiegészítő két forrásfájlból, Java osztályból áll: egy, amely a paraméterező kezelőfelületet, paraméterek feldolgozását és a RapidMinerrel felhasználói felülettel való összekapcsolást biztosítja, illetve egy második osztály, mely magát a műveletet és rekurzív algoritmust tartalmazza.

5.1. RapidMiner adatszerkezetek és objektumai

A két osztály kódjában sok helyen jelennek meg hivatkozások és referenciák a RapidMiner által használt adatszerkezetekre és objektumokra [11]. Az algoritmus megtekintése előtt érdemes az elején tisztázni ezeket.

ExampleSet

Eredményhalmazok, táblázatok. Sorai Example, oszlopoi Attribute osztályok. Ez csak egy nézet, az igazi memóriában tárolt tábla az értékekkel az ExampleTable osztály.

Example

Maguk a sorok, rekordok, melyek az értékeket tartalmazzák. Az oszloponkénti értékeire az Attribute objektumokon keresztül lehet hivatkozni.

Attributes

Gyűjtőosztály, mely egy adott ExampleSet összes attribútumát tartalmazza Attribute objektumokként. Akár új attribútumok is adhatók hozzá, átnevezhetők és törölhetők.

Attribute

Maga az attribútum objektum, melyen keresztül az Example adott oszlopának az értéke érhető el.

Szerepei a következők lehetnek

- **ID** (azonosítók): a sorok jelöléseire vannak és nem kerülnek számításra
- **Label** (magyarázandó attribútum): amikor RapidMinerben tanuló modellek kerülnek alkalmazásra, akkor ezeket a magyarázó attribútumokra határozza meg a felépítendő modellt
- **Regular** (reguláris attribútum): rendes értékek, melyeken számítások végezhetők

Az attribútumok továbbá lehetnek **nominális** típusúak (Nominal), melyek az értékeket szöveges String változókbán tartalmazzák, valamint **numerikus** értékűek (Numeric), ezekben viszont valós számok, Double értékek szerepelnek.

ExampleTable

Maga a memóriában tárolt tábla objektum, melyre az ExampleSet nézete épül. Közvetlenül ennek az értékeit állítani veszélyes és csak ExampleSet objektumon keresztül ajánlott.

InputPort

Operátor bemenetének osztálya, innen olvashatók be akár nyersadatok ExampleSet-ként esetleg itt vehetők át korábbi táblázatok.

OutputPort

Feldolgozott vagy új ExampleSet továbbadása a kimenetre vagy továbbadás következő operátornak.

5.2. Kezelőfelület

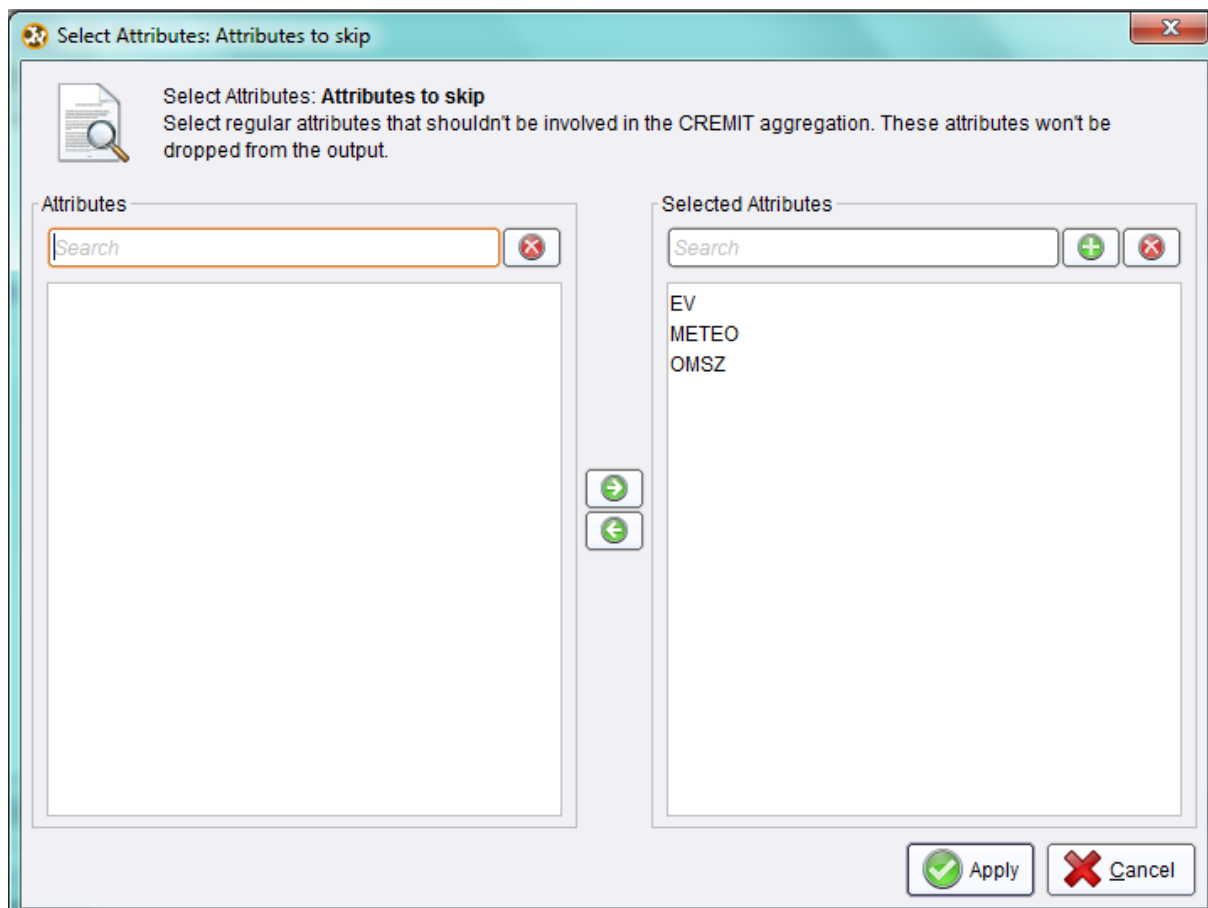
A paramétereket egy grafikus panelen lehet megadni a RapidMiner szoftveren belül, mely a következő ábrán látható.

Apply CReMIT Aggregation	
Attributes to skip	Select Attributes...
Starting time unit	1
Ending time unit	12
Maximum time shifting	2
Ending record type	actual time records
Maximum window size	6
Type of aggregation	sum

7. ábra – Az elkészített CReMIT Extension-ben található operátor kezelőfelülete.

Attributes to skip

Itt lehet kiválasztani, mely reguláris attribútumokat ne vegye be az aggregációba („ID” és „Label” típusú attribútumokat eleve nem veszi be). Ezeket az oszlopokat a számításba nem veszi be, viszont a kimenetben megmaradnak. Például szeretném megtartani az OMSZ, EV és METEO oszlopokat, de a CReMIT aggregációba nem bevonni azokat.



8. ábra – Az Attributes to skip egy új ablakot nyit, melyen megadhatjuk, hogy mely attribútumokat ne vegye bele az aggregációba. A példában az évszámokat, meteorológiai adat típusát és az OMSZ azonosítók kerültek kijelölésre.

Starting time unit

Hányadik meghagyott reguláris attribútumtól induljon az aggregálás. Jelen esetben januártól kell indulnia, ezért 1-et adok meg, és ennyit is lehet megadni legalább. Az eredeti CReMIT definícióban ez a Kezdő_hónap paraméternek felelne meg.

Ending time unit

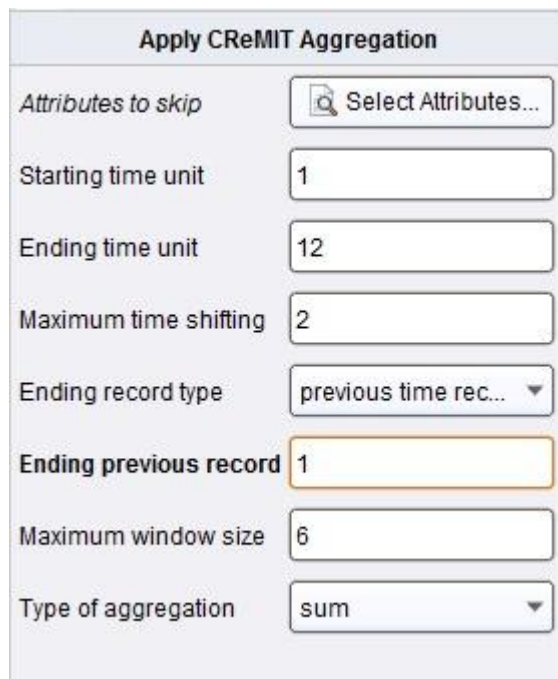
Hányadik meghagyott reguláris attribútumig tartson az aggregálás. Jelen esetben decemberig kell tartania, ezért 12-t adok meg. Az eredeti CReMIT definícióban ez a Befejező_hónap paraméternek felelne meg.

Maximum time shifting

Hány sorral előrébb induljon el az aggregáció, mi adataink esetén hány évvel. Ha arra van szükség, hogy csak aktuális sorban történjen az aggregáció, akkor ide 0 értéket kell megadni. Jelen esetben két évvel ezelőtről kell indulnia, ezért 2-t adok meg. Egynél több sornyi visszalépés esetén számokkal történik az előző sor jelzése („2p”, „3p”), kivéve egy visszalépés esetén, ott marad sima „p”. Az eredeti CReMIT definícióban ez a Kezdő_melyik_év paraméternek felelne meg.

Ending record type

Itt lehet kiválasztani, hogy a befejező attribútum az előző sorok valamelyikében vagy az aktuális sorban szerepeljen. Előző sorok esetén egy új paramétertípus jelenik meg, ami az ábrán is látható.



Apply CReMIT Aggregation	
Attributes to skip	Select Attributes...
Starting time unit	1
Ending time unit	12
Maximum time shifting	2
Ending record type	previous time rec...
Ending previous record	1
Maximum window size	6
Type of aggregation	sum

9. ábra – Extra parameter az Ending record type átállítása után. Ha az ábrán látható módon vannak beállítva a paraméterek, akkor 2p1-től indul és p12-ig tart az aggregáció.

Ending previous record

Hányadik sor legyen az, ahol az aggregáció befejeződjön. Az eredeti CReMIT definícióban ez a Befejező_melyik_év paraméternek felelne meg.

Maximum window size

Maximális ablakméret megadása, hogy egy ablakba hány hónapot, időegységet, értéket vonjon be, mielőtt új ablakot hozna létre. 1-es ablakméret esetén nem történnek számítások. Az eredeti CReMIT definícióban ez a Max_ablakméret paraméternek felelne meg.

Type of aggregation

Az aggregáció típusa, melyet a módszer az ablakokba bevont értékeken hajt végre. Ez egy legördülő lista, melyből a következőket lehet kiválasztani:

- sum – összegzés
- avg – átlagolás
- max – maximum
- min – minimum

Az eredeti CReMIT definícióban ez a Művelet paraméternek felelne meg.

Az eredeti CReMIT definícióhoz képest az operátorban nem lettek definiálva a Kezdő_év és Befejező_év paraméterek. A felhasználó egy filterező RapidMiner operátorral meg tudja adni, mely sorokra van szükség és melyekre nem.

5.3. Operátor osztály

Ez az osztály a beépített RapidMiner Operator absztrakt osztályából örököl, melynek *getParameterTypes* függvényét és *doWork* metódusát felül kell definiálni, ezek tartalma ugyanis az absztrakt osztályban üresek. Az egyéb szolgáltatásokat, amibe beletartoznak az operátor XML fájlban meghatározott leírásának elérése, a RapidMinerben naplózás, a folyamat elérése és egyéb rejtett funkcionálisok már előre meg vannak írva, így új operátor létrehozásakor már nem kell ezekre koncentrálni.

Ez fogja nyújtani az összeköttetést a felhasználóval, a RapidMiner elemző folyamatával és az általunk megírt CReMIT módszert tartalmazó logikával. Az osztályban először definiálni kell a használandó paramétereket, bemeneti és kimeneti portokat, *getParameterTypes* függvény felüldefiníálásával összeállítani a kezelőfelületet, *doWork* metódusban pedig végrehajtani az operátor futását.

Változók deklarálása

A paramétereket publikus konstansokként kellett definiálnom. Elfogadott szokás a Java elnevezési konvenciókban, hogy a konstans változók nevei mindig nagy betűkből állnak. Szöveges értékeik a paraméterek nevei lesznek, a konstansok változóneveire pedig „kulcsként” fog tudni hivatkozni az operátor paraméterek kezelőfelületéhez adáskor, valamint paraméterértékek lekérdezésekor.

```
public class CcreditAggregationOperator extends Operator {

    public static final String PARAMETER_ATTRIBUTES_TO_SKIP = „Attributes to skip”;
    public static final String PARAMETER_START_UNIT = "Starting time unit";
    public static final String PARAMETER_END_UNIT = "Ending time unit";
    public static final String PARAMETER_MAXIMUM_TIME_SHIFTING =
        "Maximum time shifting";
    public static final String PARAMETER_UNTIL_TYPE = "Ending record type";
    public static final String PARAMETER_UNTIL_RECORD = "Ending previous record";
    public static final String PARAMETER_MAX_WINDOW_SIZE = "Maximum window size";
    public static final String PARAMETER_AGGREGATION_TYPE = "Type of aggregation";
    public static final String PARAMETER_LEAVE_ORIGINAL = "Leave original";
```

Itt megadtam, hogy milyen listaelemeket tartalmazzon a Type of Aggregation és Ending Record Type paraméterek legördülő listái.

```
    private final String[] OPERATIONS = { "sum", "avg", "max", "min" };

    private final String[] PREVIOUS_OR_ACTUAL =
        { "previous time records", "actual time records" };
```

A bemeneti és kimeneti portok létrehozásra kerülnek. Itt meg kellett adnom azt is, hogy ExampleSet típusú objektumot vár bemenetnek és kimenetnek is ezt adja tovább.

```

private InputPort exampleSetInput =
    getInputPorts().createPort("example set", ExampleSet.class);

private OutputPort exampleSetOutput =
    getOutputPorts().createPort("example set");

```

Miután létrehoztam a paraméterek és a portok változóit, két listát deklaráltam. A Java nyelvi elemeibe tartozó List adatszerkezet nem más, mint egy tároló, mely méretét nem kell megadni, végtelen mennyiségű elem tárolható és kivehető belőle. Az algoritmusban leginkább kihasznált előnyük viszont az, hogy ha a programozó a lista legelejére, a 0. indexbe szűr be elemet, akkor az összes többi eltolja eggyel jobbra, helyet adva az új elemnek, ehhez pedig nem kell külön logikát írnia.

A *timeRecords* lista sorokat, azaz Example objektumokat fog tárolni, míg a *keptAttributes* listába azok a reguláris attribútumok fognak később bekerülni, melyeken számítási műveleteket kell végrehajtani, azaz nem kerültek szűrésre paraméterezés közben. A sorok eltárolására azért van szükség, mert így egyszerűbb és biztonságosabb például egyszerre két évet bevonni a tárolóba, és az aggregáló műveletbe bevonni az előző év és az aktuális év hónapjainak meteorológiai értékeit.

```

private List<Example> timeRecords;
private List<Attribute> keptAttributes;

```

Paraméterező felület létrehozása – `getParamterTypes` függvény

A `getParameterTypes` függvény egy List típusú objektumot ad visszatérési értéként, mely a `ParameterType` osztályú paramétereket tartalmazza. Így itt nem lehet különösebb logikát írni, csak sorba hozzáadni elemenként egy listába az új paramétereket. A paraméterek egy *types* nevű listába fognak kerülni és hozzáadásnál polimorfizmus lesz jelen, ugyanis adhatók `ParameterTypeInt` (egész szám), `ParameterTypeCategory` (legördülő lista) és egyéb paraméter objektumok.

```

@Override
public List<ParameterType> getParameterTypes() {
    List<ParameterType> types = super.getParameterTypes();

```

A paraméterek létrehozásakor és hozzáadásukkor rövid leírást is adok, valamint alapértelmezett értékeket is beállíthatok. Az Ending record type paraméter megkapja a két listaelemet tartalmazó *PREVIOUS_OR_ACTUAL* konstans tömböt. Az operátor logikájában a lista elemek egész számokként vannak reprezentálva, melyek 0-tól indulnak, így ha arra van szükség, hogy az alapértelmezetten kiválasztott listaelem a második legyen, akkor ennek 1 értéket kell megadni.

```

types.add(new ParameterTypeAttributes(PARAMETER_ATTRIBUTES_TO_SKIP, "Select
regular attributes that shouldn't be involved in the CREMIT aggregation.
These attributes won't be dropped from the output.", exampleSetInput,
true));

types.add(new ParameterTypeInt(PARAMETER_START_UNIT, "Index of time
attribute to start the aggregation from.", 0, Integer.MAX_VALUE, 1, false));

types.add(new ParameterTypeInt(PARAMETER_END_UNIT, "Index of time attribute
where CREMIT aggregation should end.", 0, Integer.MAX_VALUE, 12, false));

```

```
types.add(new ParameterTypeInt(PARAMETER_MAXIMUM_TIME_SHIFTING, "Amount of
time shifting and number of previous records to involve. Set to zero if you
only want to work with actual records.", 0, Integer.MAX_VALUE, 1, false));

types.add(new ParameterTypeCategory(PARAMETER_UNTIL_TYPE, "",
PREVIOUS_OR_ACTUAL, 1, false));
```

A paraméterekhez rendelhetők függőségek is a *registerDependencyCondition* metódussal. Így a következő paraméter elrejtethető a felhasználó elől attól függően mi van az Ending record type legördülő listában kiválasztva.

```
ParameterType type = new ParameterTypeInt(PARAMETER_UNTIL_RECORD, "", 1,
Integer.MAX_VALUE, 1, false);

type.registerDependencyCondition(new EqualTypeCondition(this,
PARAMETER_UNTIL_TYPE, PREVIOUS_OR_ACTUAL, true, 0));
types.add(type);
```

Majd a maradék két paramétert is elhelyezem a felületen. Fontos a sorrend, ezért is került ez az utolsó kettő a végére. A művelet típusának kiválasztásához az előbb deklarált listaelemeket tartalmazó *OPERATIONS* konstans tömböt adom hozzá, alapértelmezett listaelem pedig a legelső 0 indexű lesz.

```
types.add(new ParameterTypeInt(PARAMETER_MAX_WINDOW_SIZE, "Maximum number of
values of time attributes to involve in one window.", 1, Integer.MAX_VALUE,
6, false));

types.add(new ParameterTypeCategory(PARAMETER_AGGREGATION_TYPE, "Type of
aggregation to apply on involved values in the windows.", OPERATIONS, 0,
false));
```

Végül visszatérési értéknek a regisztrált paramétereket tartalmazó listát adja vissza a *getParameterTypes* függvény. Nem kellett foglalkoznom azzal, hogy mely metódusnak van szüksége a paramétereket tartalmazó listára, ez az *Operator* absztrakt osztályban már meg van írva.

```
return types;
} // függvény vége
```

Az operátor végrehajtása – *doWork* metódus

Minden *RapidMiner* operátor rendelkezik *doWork* metódussal, melyet felül kell definiálni és az operátor elemző logikáját kell tartalmaznia. A metódus akkor fut le, amikor az elemzési folyamat futás közben ehhez az operátorhoz ér. Ez az operátor eltárolja a bemeneti táblázatot a bemeneti portról, beállítja a paraméterek értékeit a kezelőfelületen megadottak alapján. Ezekután végighalad az összes soron, majd soronként végrehajtja a CReMIT aggregációt az egész *ExampleSet* objektumon.

A metódus definiálása után egy új *result* *ExampleSet* objektumba tárolja el a bemeneti port tartalmát, mely a port *getData* metódusával érhető el. Az operátor működése során végig ezen az eltárolt *ExampleSet*-en fog dolgozni és ezt is fogja kimenetként továbbadni.

```
@Override
public void doWork() throws OperatorException {

    ExampleSet result = exampleSetInput.getData();
```

Következő lépésben lekéri a paraméterek tartalmait, melyek lehetnek szövegesek vagy egész szám típusúak, majd változóba menti őket. Itt is látszik, hogy a legelején definiált paraméterkonstansok kulcsként funkcionálnak és az alapján éri el a kezelőfelületet. Az egész szám típusú *until* változó esetén azt is ellenőriznie kell, hogy az Ending record type legördülő listájában mely kategória volt kiválasztva.

```
String attributesToSkip =
    getParameterAsString(PARAMETER_ATTRIBUTES_TO_SKIP);

int startUnit = getParameterAsInt(PARAMETER_START_UNIT);
int endUnit = getParameterAsInt(PARAMETER_END_UNIT);

int maximumTimeShifting =
    getParameterAsInt(PARAMETER_MAXIMUM_TIME_SHIFTING);

int until;
if(getParameterAsInt(PARAMETER_UNTIL_TYPE) == 1)
    until = 0;
else
    until = getParameterAsInt(PARAMETER_UNTIL_RECORD);

int maxWindowSize = getParameterAsInt(PARAMETER_MAX_WINDOW_SIZE);
int aggregationType = getParameterAsInt(PARAMETER_AGGREGATION_TYPE);
```

A paraméterek lekérésénél külön hibakezeléssel nem kellett foglalkoznom, hiszen az elején korlátoztam a felhasználó beviteli lehetőségeit. Ha ez sikerült, akkor referenciát ad a rekurzív elemző logikát tartalmazó osztályra, iterátorokat hoz létre a bemeneti tábla soraira és reguláris attribútumaira. Az iterátorokra azért van szükség, mivel a RapidMiner szakemberei a RapidMiner Java API dokumentációjában nem garantálják biztonságosnak az egyszerű for-ciklusokkal való dolgozást közvetlen az ExampleSet objektumokon, ezért a RapidMiner API beépített iterátor objektumait javasolják külön Example objektumokra (ExampleReader) és attribútumokra (AttributeIterator). Az attribútumok iterátorára csak az elején lesz szükség, a referenciák utána ugyanis lista objektumokban lesznek, míg a sorokét aktívan fogja használni az operátor. Példányosításra kerülnek még az üres *timeRecords* és *keptAttributes* listák, hogy lehessen bennük elemeket tárolni.

```
CrematAggregation cremat = new CrematAggregation();

ExampleReader exampleReader = (ExampleReader) result.iterator();
AttributeIterator attributeIterator =
    (AttributeIterator) result.getAttributes().iterator();

timeRecords = new ArrayList<Example>();
keptAttributes = new ArrayList<Attribute>();
```

A reguláris attribútumok szűrését a következő lépésekkel végzi: először az iterátor segítségével az összes reguláris attribútumot a *keptAttributes* listába tárolja el, majd ha paraméterrel definiálva voltak elhagyandó attribútumok, akkor végighalad a megadott elhagyandó attribútumokon és megvizsgálja, hogy szerepelnek-e a megtartott attribútumok listájában. Ha igen, akkor eldobásra kerülnek a keptAttributes listából és továbbhalad.

Az elhagyandó attribútumok listáját pipeline-karakterekkel elválasztott szöveggént adja vissza az `Attributes` to skip paraméter, például EV, METEO és OMSZ kiválasztása esetén „EV | METEO | OMSZ”. A Java nyelvi elemeibe épített *split* függvénnyel egyszerű összefüggő szöveget elválasztó karakterek és reguláris kifejezések alapján több sztring elemet tartalmazó tömbre bontja, melyen `foreach` ciklussal végigiterál.

```
while (attributeIterator.hasNext()) {
    keptAttributes.add(attributeIterator.next());
}

if(attributesToSkip != null && attributesToSkip.length() != 0) {
    for (String attributeName : attributesToSkip.split("\\|")) {
        if(keptAttributes.contains(result.getAttributes()
            .get(attributeName))) {
            keptAttributes.remove(result.getAttributes().
                get(attributeName));
        }
    }
}
```

Ha az előkészületek megvoltak, akkor elkezdődhet a sorok iterátorának léptetése egyenként, *timeRecords* listához adás és a CReMIT aggregációk végrehajtása aktuális soronként. Itt jön képbe a `List` objektumok egyik előnye, vagyis a 0. indexre beszúrva új `Example` objektumot a többi benne maradt `Example` automatikusan eltolódik jobbra, eggyel nagyobb indexekre, hogy a lista elején legyen hely az új elemnek.

A listához adás után egy ellenőrzés van, mely nem enged meg több sort tartalmazni a listában, mint amennyit egy aktuális sor aggregáló műveletében felhasználunk. Például ha a kezdő ablak két évvel ezelőtti, a befejező ablak pedig az aktuális évben van, akkor a lista elemei 0, 1, 2 indexekre a következők:

0. Aktuális év sora
1. Tavalyi év sora
2. Kettővel ezelőtti év sora

Tehát a listában háromnál több elem ilyenkor nem lehet. Amikor új aktuális sor kerül be a lista elejére, akkor a 3. indexű sort már eldobja a listából.

Végül a bevont sorokon meghívja a CReMIT logikában található *aggregateStart* metódust. Ez paraméterként megkapja a *result* `ExampleSet` objektumot, amin dolgozik – ne felejtjük el, hogy bár Java nyelven sosincs referencia szerinti függvényparaméter átadás csak értékbeli, a Java környezet minden objektumot referenciaként kezel és ekkor a referencia értékét adja át a függvénynek. Ezért módosul a tábla akkor is, ha más osztály metódusában végez rajta műveletet – a bevont sorok és megtartott attribútumok listáját, valamint a kezelőfelületen definiált paramétereket.

```

while (exampleReader.hasNext()) {
    timeRecords.add(0, exampleReader.next());

    if(timeRecords.size() > maximumTimeShifting + 1)
        timeRecords.remove(timeRecords.size() - 1);

    cremit.aggregateStart(result, timeRecords, keptAttributes,
        maximumTimeShifting, startUnit, endUnit, until, maxWindowSize,
        aggregationType);
}

```

A feldolgozott és előállított ExampleSet tábla végül a kimeneti portra kerül.

```

    exampleSetOutput.deliver(result);
}

```

5.4. CReMIT módszert alkalmazó rekurzív algoritmus

Ahogy említettem külön elválasztottam maga az operátor és a CReMIT eljárás kódját az átláthatóság kedvéért. Ez az osztály tartalmazza az *applyOpr* függényt, mely egy ablak összes értékén végrehajtja a négy művelet valamelyikét, az *aggregateStart* metódussal új szimpla elemből álló ablakot hoz létre, az *aggregateWindow* pedig új ablakokat hoz létre az előző szimpla ablakhoz és az utániakhoz való időegységek hozzáadásával a maximális ablakméret eléréséig.

Egy maximum ablakméret elérése után egy következő kezdő attribútumra lépés következik, újra elkezdődnek az ablaksorozatok értékeinek generálása a maximális ablakméret eléréséig és így tovább. Az *aggregateStart* önmagában hívja meg az *aggregateWindow* metódust. Bár az attribútumokon való léptetés iterációval történik mindkét metódusban, a régebbi sorokból újabb sorokra (évekre) való átlépést rekurzív metódushívásokkal oldottam meg, ami azt jelenti, hogy egy metódus önmagát hívja meg újra, esetleg más paraméterezéssel. Mind az *aggregateStart*, mind az *aggregateWindow* tartalmaz önmagukra rekurzív hívásokat.

A három metódus közül csak az *applyOpr* nem került dokumentálásra, ugyanis a benne lévő utasítások csak azt csinálják, hogy a paraméterként kapott ablak List típusú objektum elemein végrehajtja a számításokat.

Új ablaksorozat indítása – *aggregateStart* metódus

Egy aggregáció sorozatot ezzel a metódussal kezd el, ez fog létrehozni egy ablaksorozat elemeinek tárolására List objektumot és kerül be először az első szimpla ablak eleme. Függvényparaméterként rendre a következőket kapja:

- **result** – ExampleSet objektum, melyen a CReMIT eljárást végre kell hajtani.
- **exampleList** – sorok listája, melyekből az operátor aktuális iterációjában értékeket kell elérni.
- **keptAttributes** – meghagyott reguláris attribútumok listája.
- **exampleIndex** – exampleList-ben szereplő sor indexe, ahonnan az értéket kell venni.
- **start** – kezdő attribútum indexe a meghagyott reguláris attribútumok listájában. Jegyezzük meg, hogy ez nem csak a kezelőfelületen megadott kezdő attribútum indexe

lehet, hanem minden egyes rekurzív metódushíváskor eggyel megnövelve adjuk át ezt a változót az új metódushívásnak.

- **end** – befejező attribútum indexe a meghagyott reguláris attribútumok listájában, az egész elemző folyamatra nézve
- **untilExample** – exampleList-ben szereplő sor indexe, melynél tovább már ne léptessen. Triviálisnak hangzik az, hogy ha a befejező sor egy évvel ezelőtti, akkor csak a listát kell módosítani és erre nincs szükség, de értékeket mindig csak aktuális sorokba írhatunk. Ezért van szükség íráskor az aktuális sor indexére és ezért kell külön a befejező sorról információ.
- **maxWindowWidth** – maximális ablakméret
- **aggregationType** – végrehajtandó művelet sorszáma

```
public void aggregateStart(ExampleSet result,
                          List<Example> exampleList,
                          List<Attribute> keptAttributes,
                          int exampleIndex,
                          int start,
                          int end,
                          int untilExample,
                          int maxWindowWidth,
                          int aggregationType) {
```

A metódus kezdetén új változókat deklaráltam. *s* szöveges változóba fog kerülni a Kezdő_melyik_év jelzése („a”, „p”, „2p”, stb.), *newAtt* egy új beszúrandó attribútumnak a táblázat végére, *getAtt* az az attribútumra, ahonnan értéket akarunk venni az ablakba, *setAtt* pedig az az attribútum, melybe az ablakon végrehajtott művelet értékét akarjuk eltárolni.

```
String s;
Attribute newAtt;
Attribute getAtt;
Attribute setAtt;
```

Az *s* sztringet beállítja a megfelelő jelzés alapján. Ha aktuális sorban van, akkor a jelzés „a” lesz, ha egy sorral korábbanál, akkor „p”, ha egynél több, akkor a „p” jelzés elejéhez hozzáfűzi az *exampleIndex* értékét.

```
switch(exampleIndex) {
    case 0:      s= "a"; break;
    case 1:      s= "p"; break;
    default:     s= exampleIndex + "p"; break;
}
```

Egy foreach-ciklussal végigiterál az attribútumokon, amikkel számolni kell. Itt biztonságos iterátor használata nélkül léptetni, a referenciák ugyanis már benne szerepelnek a listában. A ciklus elején van egy elágazás, mely kimondja, hogy ha az aktuális sorban túllép a befejező attribútumon, akkor az egész metódus futásából lépjen ki a return utasítással. Ekkor a vezérlés visszakerül az operátor osztályának a *doWork* metódusához, mely továbbiterál a táblázat sorain, ha még nem végzett.

```
for (Attribute a : keptAttributes) {
    if(exampleIndex == untilExample &&
        keptAttributes.indexOf(a) + 1 > end)
        return;
```

A *valuesInWindow* lista valós számértékeket fog tartalmazni. Az *aggregateStart* metódusban ahogy iterál a foreach-ciklusban, ez minden ablaksorozat kezdő attribútumára felüldefiniálódik egy új listával. A lista átadásra fog kerülni az ablaksorozat kezdő egységének számértékének hozzáadása után az *aggregateWindow* metódusnak, mely további értékeket ad a listához és értékenként új ablakokat állít elő.

```
List<Double> valuesInWindow = new ArrayList<Double>();
```

Ha az ablaknak még nincs létrehozva attribútum a táblázatban, az annak nullreferenciájának vizsgálatával határozható meg. Ha az új ablak névével keres a táblázat eddigi attribútumai között és nem talál, akkor létre kell hoznia. Ehhez a RapidMiner AttributeFactory statikus metódusára van szükség, mely attribútumnév és eltárolandó érték típusának megadása után készít egy új Attribute objektumot. Az objektumot hozzáadja a táblázathoz, szerepét pedig regulárisra állítja. Ez az attribútum kezdetben a táblázat összes sorára üres. Ha valamelyik következő sorra való futásnál ez az attribútum már létezik, akkor itt nem csinálunk semmit.

```
if( result.getAttributes().get(s + a.getName()) == null ) {
    newAtt = AttributeFactory.createAttribute(s + a.getName(),
    Ontology.REAL);
    result.getExampleTable().addAttribute(newAtt);
    result.getAttributes().addRegular(newAtt);
}
```

getAtt az az attribútum lesz a táblázatból, melynél az aktuális attribútum iteráció tart a keptAttributes listában és név alapján veszi a táblából. *setAtt* az az attribútum lesz, amely sorába az értéket fogja írni. Ha egy évvel ezelőtti sorból kell értéket kivenni, de az aktuális sor a táblázat első soraként van jelen, akkor csak üres értéket tud beszúrni. Szimpla ablaknál még nem kell műveletet végezni, csak az *exampleIndex* által jelölt sorból az értéket eltárolja az ablaksorozat első elemeként a *valuesInWindow* listába és másolja az aktuális sor attribútumába az értéket. Ha eddig nem volt, akkor létrehoz egy *windowingCycle* egész szám változót 1 kezdeti értékkel, mely az *aggregateWindow* metódusban követni fogja, hogy éppen hány elemű ablaknál tart.

```
getAtt = result.getAttributes().get(a.getName());
setAtt = result.getAttributes().get(s + a.getName());

if(exampleList.size() < exampleIndex + 1) {
    exampleList.get(0).setValue(setAtt, Double.NaN);
} else {
    valuesInWindow.add(exampleList.get(exampleIndex).getValue(getAtt));
    exampleList.get(0).setValue(setAtt,
                                exampleList.get(exampleIndex).getValue(getAtt));
}

int windowingCycle = 1;
```

Az első szimpla ablak elkészítése után további ablakokat fog képezni az *aggregateWindow* metódus hívásával, mely hasonlóan működik, mint a hívó *aggregateStart*, új függvényparaméterekként viszont megkapja az ablak első ablakelemet tartalmazó listát, két paraméterként az *exampleIndex* változót, kezdő attribútum indexeként az attribútumokon való iteráció aktuális elemét és a *windowingCycle* változót. Ha az *aggregateWindow* metódus lefutott a saját rekurzív hívásaival vagy túllépte a maximális ablakméretet, akkor továbblép a kezdő attribútumokon való iteráción és az előző utasításokat újra futtatja a következőn.

```

        aggregateWindow(result, exampleList, keptAttributes, valuesInWindow,
            exampleIndex, exampleIndex, a, end, untilExample, windowingCycle,
            maxWindowWidth, aggregationType);
    } // for-ciklus vége

```

Az attribútumokon való iteráló ciklusból a kilépés feltétele az, hogy van az összes megtartott attribútumon haladjon végig, vagy ha aktuális sorban van, akkor a befejező attribútumig menjen. Ezek után a ciklusból kilépve ha a mostani sor nem egyezik meg a kezelőfelületen paraméterként megadott *untilExample* változóval (például éppen egy előző év sorából képzett ablakokat és az *untilExample* aktuális évre vonatkozik), akkor saját magát rekurzívan meghívja következő sorra lépéssel. Az 1-el való csökkentés azért kell, mert egy korábbi sornak a *timeRecords* és *exampleList* listákban mindig nagyobb számú indexszel rendelkezik, mint mondjuk az aktuális sor, melynek indexe a két listában mindig 0. A *start* függvényparaméter rekurzív híváskor mindig 1 értéket kap.

```

        if(exampleIndex > untilExample)
            aggregateStart(result, exampleList, keptAttributes, (exampleIndex-1),
                1, end, untilExample, maxWindowWidth, aggregationType);
    }
} // metódus vége

```

Az itt meghívott *aggregateWindow* és a rekurzív hívások lefutása végén a hívás visszakerül az operátor osztályban található *doWork* metódushoz, ami tovább iterál a táblázat sorain, ha még nem ért végig.

Attribútumok bevonása egy ablakon belül – *aggregateWindow* metódus

Látszólag ugyanúgy működik, mint hívója, az *aggregateStart* metódus, ám két céllal lett létrehozva: az ablakokat a folyamatosan növekvő *valuesInWindow* listából kell képeznie, valamint kezelnie kell egy ablakban a sorok közötti átlépést, például egy „p11-a3” ablak esetén.

Az új függvényparaméterek közé tartozik a *valuesInWindow*, mely az *aggregateStart* metódusból veszi át az ugyanolyan nevű listát, az *exampleIndex2*, mely az ablak utolsó elemének sorának indexe, valamint az ablakozási ciklus változója. *exampleIndex2* nem csak a következő sor indexe lehet, hanem akár ugyanannak a sornak is, mint *exampleIndex1* sorának, például első meghíváskor egy „p1-p2” ablak esetén. A metódus elején is csak annyi változás van az előzőhöz képest, hogy létrehozásra kerül egy szöveges *s2* változó az ablak utolsó elemének jelölésére.

```

private void aggregateWindow(ExampleSet result,
    List<Example> exampleList,
    List<Attribute> keptAttributes,
    List<Double> valuesInWindow,
    int exampleIndex1,
    int exampleIndex2,
    Attribute start,
    int end,
    int untilExample,
    int windowingCycle,
    int maxWindowWidth,
    int aggregationType) {

```

```

    String s1;
    String s2;

```

```

Attribute newAtt;
Attribute setAtt;
Attribute getAtt;

switch(exampleIndex1) {
    case 0:      s1="a"; break;
    case 1:      s1="p"; break;
    default:     s1=exampleIndex1 + "p"; break;
}

switch(exampleIndex2) {
    case 0:      s2="a"; break;
    case 1:      s2="p"; break;
    default:     s2=exampleIndex2 + "p"; break;
}

```

A for-ciklusban bekerült egy változás az elejére, hogy ugyanabban a sorban ha a *start* változó által meghatározott attribútum előtt vagy ugyanannál az attribútumnál tart az iteráció, akkor *continue* utasítással léptesse tovább a ciklust. Erre azért van szükség, hogy ki legyenek küszöbölve az olyan anomáliák, mint például „a1-p12” vagy „p1-p1”. Következő ellenőrzésnél az eredetihez képest hozzákerült az is, hogy az ablakozási ciklus ne haladja meg a maximális ablakszélességet, és *return* helyett *break* utasítással szakítsa meg a ciklust, hogy a módszer többi része végrehajtásra kerüljön.

```

for (Attribute a : keptAttributes) {
    if(keptAttributes.indexOf(a) <= keptAttributes.indexOf(start) &&
        exampleIndex1 == exampleIndex2)
        continue;

    if((exampleIndex2 == untilExample &&
        keptAttributes.indexOf(a) + 1 > end) ||
        windowingCycle >= maxWindowWidth )
        break;
}

```

Létrehozza az új ablak attribútumát az *ExampleSet*-ben ha még nem létezik, bevonja a következő értéket a *valuesInWindow* listába, a listát felhasználva pedig a táblázat aktuális sorába az *applyOpr* által kiszámított érték kerül be. Az ablakozási ciklust eggyel növeli.

```

if( result.getAttributes().get(s1 + start.getName() + "-" + s2 +
a.getName()) == null) {
    newAtt = AttributeFactory.createAttribute(s1 + start.getName() + "-"
+ s2 + a.getName(), Ontology.REAL);
    result.getExampleTable().addAttribute(newAtt);
    result.getAttributes().addRegular(newAtt);
}

getAtt = result.getAttributes().get(a.getName());
setAtt = result.getAttributes().get(s1 + start.getName() + "-" + s2 +
a.getName());

if(exampleList.size() < exampleIndex1 + 1) {
    exampleList.get(0).setValue(setAtt, Double.NaN);
} else {
    valuesInWindow.add(exampleList.get(exampleIndex2).getValue(getAtt));
    exampleList.get(0).setValue(setAtt, applyOpr(valuesInWindow,
aggregationType));
}

```

```

        windowingCycle = windowingCycle + 1;
    } // for-ciklus vége

```

Az *aggregateWindow* rekurzív hívása pedig itt is alkalmazásra kerül a sorok közötti határátlépések miatt.

```

        if(exampleIndex2 > untilExample)
            aggregateWindow(result, exampleList, keptAttributes, valuesInWindow,
                            exampleIndex1, (exampleIndex2 - 1), start, end, untilExample,
                            windowingCycle, maxWindowWidth, aggregationType);
    }
} // metódus vége

```

6. Elemzési esettanulmányok

A kialakított elemzési folyamat először 2013 nyarán rovarcsapdák fogásai és klímajellemzők összefüggéseinek vizsgálatára került alkalmazására, majd 2014 elején falevé elvesztés, mortalitás klíma összefüggéseinek elemzésére használtam. A rovarfogások elemzésének feladata kisebb méretű, a másik feladat sokkal nagyobb adathalmazt jelentett. Az alapfeladat azonban mindkét esetben hasonló: az erdészeti adatokat, mint függő változókat kell összevetni a havi meteorológiai adatok és azokból a CReMIT által előállított időszakos adatokkal, mint független változókkal.

6.1. Rovarfogási adatok elemzése

A feladathoz három bemeneti nyersadat típussal rendelkezttem, az OMSZ meteorológiai mérések adataival, a rovarfogási helyek koordinátaival és 9 rovarfaj fogási adataival.

Az OMSZ meteorológiai adatok 1961-től 2010-ig terjednek, januártól decemberig. Összesen 1104 koordináta szerepel ebben a nyersadatban, melyekre OMSZ azonosítókkal lehet hivatkozni. Ezek a mérőhelyek Magyarországra vonatkozóan 1104 rácspontra helyezkednek el 0,1 fokos felbontásban, helyzetüket x és y EOVS koordináták jelölik. A mért meteorológiai adatok havi csapadékösszeg és havi középhőmérséklet.

Rovarfogási helyek bemeneti táblázatában fogási helyenként 23 település szerepel névvel megjelölve, mellettük GPS és x-y EOVS koordinátákkal. Nem minden rovarfaj esetén fordult elő, hogy minden fogási helyen sikerült annyi számú egyed fogni a fénycsapdákkal, hogy azt az összefüggés vizsgálatokba bevonásra kerüljenek, így vannak fajok, ahol nem kerül be mind a 23 fogási hely az elemzés folyamatába.

Maguk a rovarfogási adatok 9 faj miatt 9 táblázatból állnak. A táblázatok első oszlopában az évszámok szerepelnek, általában 1961-től 2012-ig, de ez nem mind a 9 faj esetén van így: amely fajoknál a fogási helyek többségénél korai években nem, vagy ismeretlenek voltak a fogási darabszámok, azoknál a fajoknál későbbi évszámtól vannak megadva a fogások. Ezek az adatok tehát változó hosszúságúak. A táblázat többi oszlopa egy-egy fogási település név, ezek az oszlopok tartalmazzák az adott évben fogott rovar egyedeinek darabszámát.

Problémát jelent az, hogy hogyan legyenek összerendelve a fogási helyek és a mérőhelyek adatai, ugyanis nem közvetlenül ugyanazon a helyen vannak. A megoldás, hogy adott fogási helyekhez a hozzá legközelebb eső n mérési hely távolságainak és mérési adatainak fordított

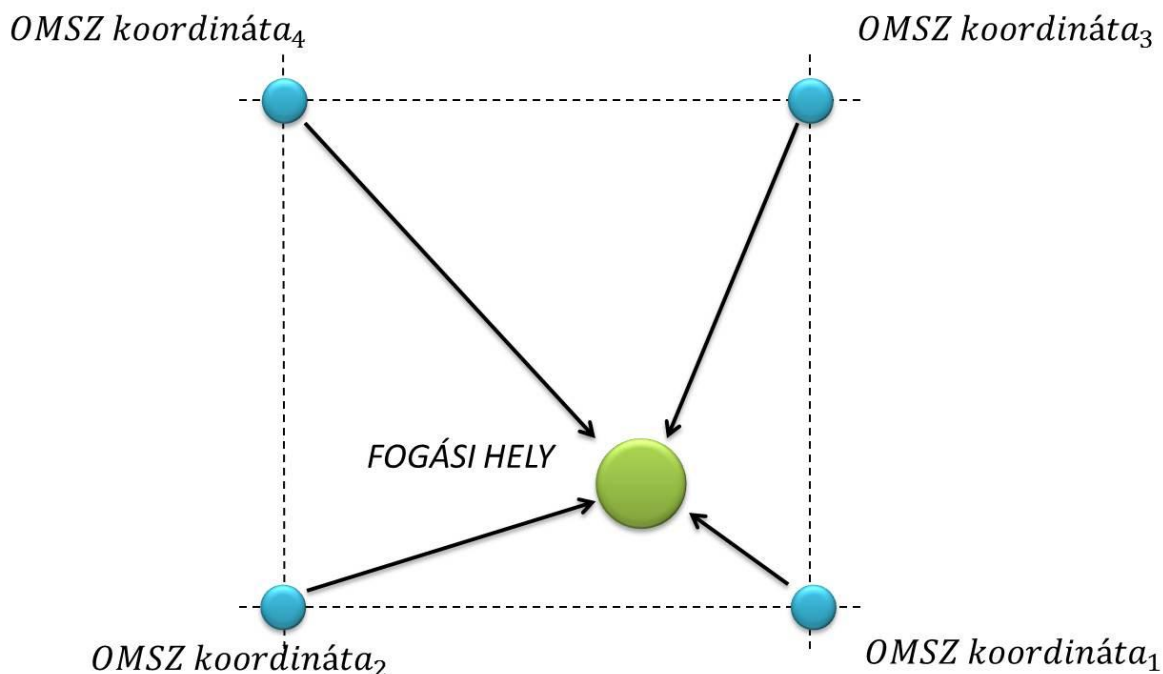
arányossággal súlyozott átlagával adom meg a meteorológiai értékeket. Egy évre és hónapra az új meteorológiai érték ezzel a módszerrel való előállításának képlete

$$\frac{\frac{1}{\text{távolság}_1} * \text{meteoadat}_1 + \frac{1}{\text{távolság}_2} * \text{meteoadat}_2 + \dots + \frac{1}{\text{távolság}_n} * \text{meteoadat}_n}{\frac{1}{\text{távolság}_1} + \frac{1}{\text{távolság}_2} + \dots + \frac{1}{\text{távolság}_n}} =$$

$$\frac{\sum_{i=1}^n \left(\frac{1}{\text{távolság}_i} * \text{meteoadat}_i \right)}{\sum_{i=1}^n \frac{1}{\text{távolság}_i}}$$

ahol n az adott fogási helyhez legközelebbi meteorológiai koordináták száma, távolság az adott fogási hely és i koordináta közötti távolság, $\text{meteorológiai adat}$ pedig adott évre és hónapra az i koordinátahoz tartozó meteorológiai adata.

Erre dolgoztam ki egy adatelőkészítő modult, melyet egy R-szkript operátor hajt végre a nyersadatok betöltése után. A szkript veszi egy fogási hely EOVS koordinátáit, majd végighalad az OMSZ mérések EOVS koordinátáin és előállítja a távolságokat tartalmazó mátrixot a fogási helyhez. A távolságok előállítása után növekvő sorrendbe rendezi távolság szerint az OMSZ azonosítókat, majd az n legkisebb távolságút választja ki és azok meteorológiai adataival, távolságaival hajtja végre a súlyozást.



2. ábra – Fogási településhez legközelebbi koordinátaához tartozó meteorológiai adat kiválasztása $n = 4$ esetén.

A távolságokat a fogási helyek és az OMSZ EOVS koordináták alapján Euklideszi távolsággal számítja ki. Ennek képlete

$$\text{távolság} = \sqrt{(\text{fogási hely}_{EOVX} - \text{mérőállomá}_{EOVX})^2 + (\text{fogási hely}_{EOVY} - \text{mérőállomá}_{EOVY})^2}.$$

Az előkészítő modul elnevezése ebben az elemzési feladatban **Euklidesz modul**. GPS koordinátákkal ellentétben az EOVS koordináták azzal az előnnyel rendelkeznek, hogy ezeket lehet a legkönnyebben felhasználni vektortávolság számítására. R programozási környezetben x és y EOVS koordinátát tartalmazó két oszlopból álló táblázatra egy egyszerű beépített paranccsal (dist) állítható elő az Euklideszi távolságokat tartalmazó mátrix. A távolságmátrixból sorba rendezés után kiválasztja a modul az n legkisebb távolságot és hozzá tartozó EOVS koordinátákat, majd leszűri az eredeti OMSZ nyersadatot, hogy csak a legközelebbi OMSZ azonosítók maradjanak meg. Ezután az új meteorológiai értékeket ebből a szűrt OMSZ táblázatból állítja elő. A feladat során a fogási helyhez 4 legközelebbi koordinátát vontam be, így n paramétere 4.

A „4. Elemzési folyamat automatizálása” nagyfejezetben látható, hogy a nyers adatok betöltése után az egész folyamat fogási településenként automatizálva van csapadékmennyiségre, középhőmérsékletre, outlierszűrés nélküli és IQR módszerrel szűrt változatokra. Az automatizáción belüli alfolyamatba beletartozik az Euklidesz modul is, hiszen ez választja ki, hogy most a csapadék vagy hőmérsékleti adatokra kell előállítani a súlyozott átlagolásokat. Ezután a CREMIT modul előállítja az időszakos adatokat és az Outlier modullal kiszűri IQR módszerrel a kiugró fogási adatokat, ha szűrt eredményekre van szükség.

Az elemzésekhez a kétváltozós lineáris regressziót és szignifikáns összefüggések kiválasztását használtam fel. A szignifikáns kiválasztásokkal külön-külön településenként össze vannak gyűjtve a szignifikáns időszakok három szignifikancia szintre: $\alpha = 0,1$ -re, $\alpha = 0,05$ -re és $\alpha = 0,01$ -re. Egy fajon belül pedig az összes fogási település korrelációs sorai automatikusan egy táblázatban vannak összefűzve. Ilyen összefűzött táblából négyet állít elő a rendszer, egyet, ami az összes r -értéket tartalmazza, a maradék háromban pedig az előbb említett három szignifikancia szint szerint szűrt változatok, ahol az eldobott értékek helyén üres cellák vannak. Az összefűzés és azon belüli szűrés segíti az átláthatóságot és az összefüggések könnyebb megtalálhatóságát.

5. táblázat – Településenkénti korrelációs sorok egy összefűzött Excel táblázatban, szignifikáns r értékek szűrése nélkül.

Település	p1	p1-p2	p1-p3	p1-p4	p1-p5	p1-p6	p2	p2-p3	p2-p4	p2-p5	p2-p6	p2-p7
FOGÁSI_HELY1	0,070	0,129	0,116	0,091	0,143	0,131	0,122	0,112	0,078	0,136	0,120	0,193
FOGÁSI_HELY2	0,079	0,136	0,097	0,073	0,125	0,101	0,120	0,080	0,053	0,110	0,086	0,206
FOGÁSI_HELY3	0,152	0,102	0,066	0,047	0,023	0,057	0,009	-0,014	-0,020	-0,027	0,017	-0,017
FOGÁSI_HELY4	-0,079	0,044	0,076	0,053	-0,084	-0,092	0,168	0,151	0,101	-0,066	-0,078	-0,132
FOGÁSI_HELY5	-0,032	0,093	0,134	0,246	0,245	0,240	0,150	0,188	0,322	0,302	0,276	0,232
FOGÁSI_HELY6	0,163	0,248	0,209	0,129	0,168	0,122	0,178	0,146	0,047	0,115	0,074	0,070
FOGÁSI_HELY7	-0,017	0,230	0,246	0,131	-0,009	0,106	0,327	0,312	0,166	-0,003	0,117	0,005
FOGÁSI_HELY8	-0,026	0,051	0,103	0,228	0,148	0,128	0,095	0,152	0,283	0,181	0,150	0,133
FOGÁSI_HELY9	0,084	0,111	0,012	0,085	-0,040	-0,093	0,058	-0,066	0,035	-0,093	-0,143	-0,116
FOGÁSI_HELY10	0,046	-0,025	-0,144	-0,231	-0,188	-0,148	-0,085	-0,202	-0,296	-0,233	-0,173	-0,224
FOGÁSI_HELY11	0,273	0,321	0,149	-0,023	-0,065	-0,094	0,163	-0,033	-0,191	-0,188	-0,184	-0,128
FOGÁSI_HELY12	0,120	0,057	-0,078	-0,105	-0,223	-0,233	-0,030	-0,170	-0,195	-0,307	-0,304	-0,360
FOGÁSI_HELY13	-0,082	0,140	0,123	0,090	-0,016	0,153	0,322	0,213	0,177	0,013	0,190	0,039
FOGÁSI_HELY14	-0,009	0,064	0,075	0,223	0,091	0,050	0,117	0,113	0,277	0,110	0,060	0,020
FOGÁSI_HELY15	0,045	0,318	0,205	0,210	0,101	0,058	0,380	0,218	0,209	0,095	0,046	-0,009
FOGÁSI_HELY16	-0,027	0,189	0,384	0,595	0,431	0,359	0,272	0,449	0,633	0,474	0,371	0,296
FOGÁSI_HELY17	0,183	0,178	0,097	0,033	-0,042	-0,131	0,099	0,022	-0,051	-0,122	-0,190	-0,219
FOGÁSI_HELY18	0,713	0,534	0,480	0,256	0,125	-0,020	0,264	0,259	0,043	-0,075	-0,167	-0,062
FOGÁSI_HELY19	0,116	0,241	-0,001	-0,001	0,074	-0,029	0,217	-0,051	-0,044	0,042	-0,061	-0,055

Szignifikánsan szűrt összefűzésekben ahol egy ablakban egyik fogási hely esetén sem volt szignifikáns korreláció, ott az ablak egész oszlopa eldobásra kerül.

6. Táblázat – Településenkénti korrelációs sorok egy összefűzött táblázatban, $\alpha = 0,05$ szignifikancia szinten szűrve. Amely ablakoknál nem volt szignifikáns r érték egyetlen település esetén sem, azok az ablakok oszlopai elhagyásra kerültek.

Település	p1	p1-p2	p1-p4	p2	p2-p3	p2-p4	p2-p5	p3-p4	p7	p7-p9
FOGÁSI_HELY1										
FOGÁSI_HELY2									0,324	
FOGÁSI_HELY3										
FOGÁSI_HELY4										
FOGÁSI_HELY5						0,322	0,302			
FOGÁSI_HELY6										
FOGÁSI_HELY7				0,327	0,312					
FOGÁSI_HELY8								0,315		
FOGÁSI_HELY9										
FOGÁSI_HELY10										
FOGÁSI_HELY11								-0,353		
FOGÁSI_HELY12										
FOGÁSI_HELY13										-0,609
FOGÁSI_HELY14										
FOGÁSI_HELY15										
FOGÁSI_HELY16			0,595			0,633	0,474	0,632		
FOGÁSI_HELY17										
FOGÁSI_HELY18	0,713	0,534								
FOGÁSI_HELY19										

6.2. Levélvesztési adatok elemzése

A folyamat következő sikeres alkalmazása falevé elvesztés és klímaváltozás összefüggés vizsgálatára történt 2014 elején. A feladathoz két bemeneti nyersadat típussal rendelkeztem, ezek közül a korábbi elemzési feladtból került felhasználásra az OMSZ meteorológiai mérési adatok havi csapadék és hőmérséklet adatai, a 13 fajcsoportot számláló levélvesztési adatokhoz pedig két segédtáblázat tartozik.

A másik táblázat évekre bontva 1990-től 2010-ig tartalmazza az egyes faegyedek levélvesztéseinek és a mortalitások mértékét. Az összes faegyed a rovarfogási adatokkal ellentétben most egy közös táblázatban szerepel. Az egyedek 6 magyarországi természetvédelmi nagytájba, ezeken belül pedig 5-5 magassági osztályba voltak sorolva, az elemzések során viszont ezek közül csak az első 3-ra volt szükség. A faegyedek száma fajcsoportonként, nagytájanként és magassági osztályonként változó mennyiségűek, valamint sok helyen találhatóak elhalt faegyedek is. A levélvesztések mértéke az LVESZ attribútumban 0-tól 9-ig terjedő egész számokkal vannak osztályozva, ugyanebben a mezőben X-el van jelölve, ha egy faegyed elpusztult.

Az egyedek fajai egyszerre vannak megjelölve szám- és betűazonosítókkal és körülbelül 150 fajba vannak besorolva. A 150 faj összesen 13 csoportba van sorolva, de ezek nincsenek jelölve a levélvesztési adatokat tartalmazó táblázatban, így szükség volt egy külön segédtáblára, mely azt mutatja, hogy melyik faj mely csoportba tartozik a 13 közül. Az egyedek földrajzi helyzetei nem EOV, hanem UTM koordinátákban voltak meghatározva, ezért egy másik kapott segédtáblázatban koordináták és az OMSZ azonosítók szerepeltek.

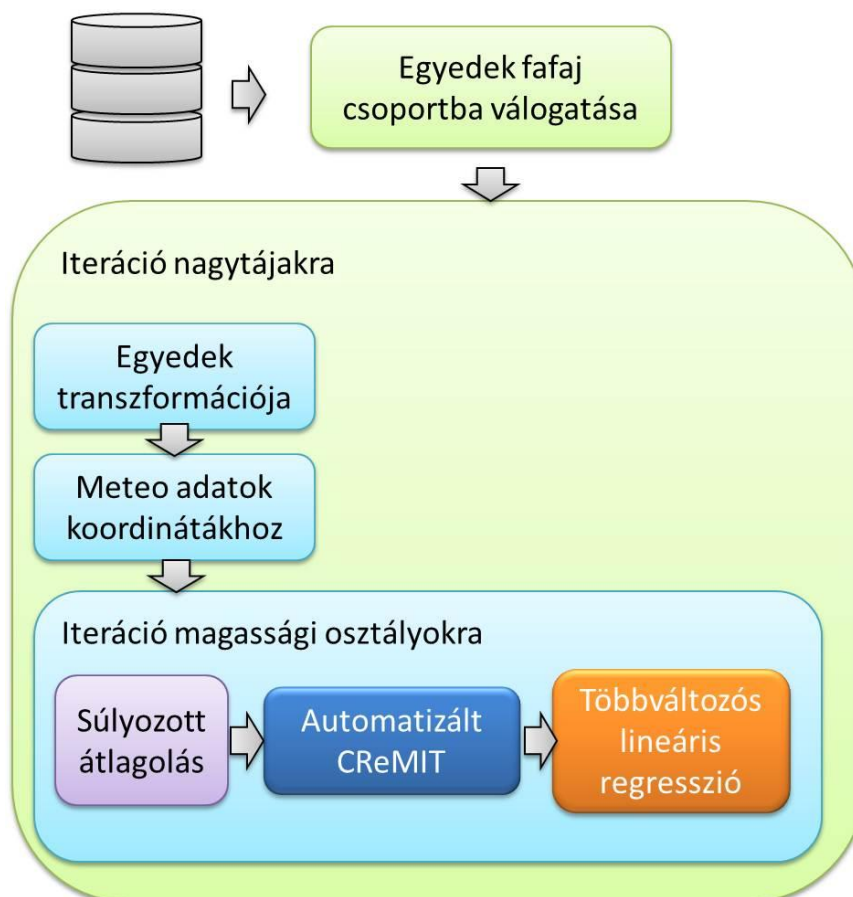
Az egyedeket fajaj csoportokba, természetvédelmi nagytájakba és 5 helyett 3 magassági osztályokba kellett csoportosítani. Ezzel $13 \times 6 \times 3$, azaz 234 féle kimeneti csoportot kaptam. A kiválogatás után a problémák között szerepel az, hogy egy kimeneti csoporton belül a

csapadék és hőmérsékleti adatok egységesek legyenek, legyen a kimeneti csoporton belül egyedek levélvesztés mennyisége és darabszáma szerint egy átlag, valamint figyelembe kell venni az elpusztult faegyedeket is.

A nyersadatok betöltése után a folyamat az összes egyedet tartalmazó bemenetből ki válogatja azokat az egyedeket, melyek fafajai beletartoznak az általam kiválasztott csoportba. Ezután nagytájanként iterál egy ciklusban és minden koordinátára előállítja az egyedek darabszámát, az évi levélvesztés átlagát, valamint az elhalt egyedek évi darabszámát. Lényeg, hogy ugyanazon koordinátával rendelkező több egyedből csak egy-egy koordináta összesített levélvesztési adatai legyenek, melyekhez később a koordináták alapján hozzárendeli az OMSZ-hez tartozó meteorológiai adatokat.

A koordinátánkénti meteorológiai adatok előállítása után a nagytáj ciklusán belül maradva a magassági osztályokra iterál egy beágyazott ciklusban. A magassági osztály ciklusán belül pedig faegyedek darabszámával súlyozott átlagolással állítja elő adott nagytájra és magassági osztályra a faj csoport évi egységes levélvesztés mennyiségeit és a meteorológiai adatokat. A beágyazott magassági osztály-ciklusban maradva pedig végrehajtja a CReMIT aggregációt, a kiugró adatok szűrését, kétváltozós lineáris regressziós vizsgálatot, valamint itt került először alkalmazásra a többváltozós lineáris regressziós vizsgálat.

A „4. Elemzési folyamat automatizálása” nagyfejezetben látott módon az CReMIT, Outlier és kétváltozós elemző modulok egy alfolyamaton belül vannak automatizálva csapadék- és hőmérséklet adatokra outlier szűrés és IQR módszerrel való szűréssel. A többváltozós



11. ábra – Elemzések futása levélvesztési adatok esetén.

vizsgálatok viszont egy külön blokkba kerülnek a CReMIT automatizáció után, ugyanis egyszerre van szüksége a csapadék- és hőmérséklet időszakos adataira outlier szűrés nélkül.

A nagytájon és azon belül magassági osztályon belüli évi levélvesztések és havi meteorológiai adatok koordinátakénti egyed darabszámokkal való súlyozásának képlete

$$\frac{\sum_{i=1}^n (adat_i * (darab_i - elpusztult_i))}{\sum_{i=1}^n (darab_i - elpusztult_i)}$$

ahol n a koordináták száma, $adat$ behelyettesíthető levélvesztés mennyiségére egy adott évben, havi csapadékmennyiségre vagy középhőmérsékletre egy koordinátára, $darab$ az faegyedek száma adott évben és $elpusztult$ az elpusztult faegyedek száma adott évben.

CReMIT által előállított időszakos adatok alapján végül a többváltozós vizsgálatokkal egyszerre lehet azt látni, hogy mely ablakok csapadék vagy hőmérséklet adatai voltak nagyobb hatással a levélvesztések mennyiségére, figyelembe véve a multikollinearitást.

7. Tapasztalatok, továbblépési lehetőségek

Feladatom végrehajtása során sikeresen értem el eredményeket a fejlesztések implementálásában statisztikai- és programozási képességeim segítségével. További RapidMiner funkcionalitások ismeretére és R programozási ismeretekre tettem szert, miközben a többváltozós regresszió elméletébe is betekinttem. A CReMIT Extension fejlesztésekor mélyebben megismertem a RapidMiner belső működését és programozhatósági lehetőségeit.

Létrehoztam egy elemzési folyamatot, mely alkalmazza a CReMIT módszer segítségével az időszakos ablakok létrehozását és időben eltolt hatások vizsgálatát. Moduláris felépítésének révén könnyű a továbbfejlesztés és egyes folyamatbeli komponensek cseréje. Az elemzések automatizálásával sikerült elérni, hogy a felhasználónak kevesebbet kelljen gépelni, ne kelljen egy-egy szimpla elemzés közben várnia arra, hogy paramétereket írjon át, ezáltal az elemzések folyamata kényelmesebbé és gyorsabbá vált. Az automatizálás közben újabb táblázatok állnak elő, melyekben egy helyen lehet látni például rovarfajok esetén a vizsgált faj összes fogási településére a korrelációs együtthatókat.

Megismerkedtem a többváltozós regresszió elemzés alapjaival, a fellépő nehézségekkel (multikollinearitás) és annak kezelési lehetőségeivel. A többváltozós regresszió vizsgálata során egyszerre lehet vizsgálni a fogási adatok, csapadékmennyiség és hőmérséklet kapcsolatát, külön „fogási adat-csapadék” és „fogási adat-hőmérséklet” helyett.

A CReMIT Extension kiegészítő elkészítésével saját magam hoztam létre RapidMiner operátort, kényelmes felületet adtam a felhasználónak paraméterek megadására, mögötte pedig strukturált logikájú rekurzív algoritmus működik. A kiegészítő kész a publikussá tételre, így bárki letöltheti és használhatja RapidMiner szoftverében.

A kialakított elemzési folyamat kapcsán látszik, hogy mennyi továbblépési pont van, legyen szó akár az elemző modulban alkalmazható eljárások fejlesztés kapcsán, vagy a CReMIT modulban alkalmazható transzformációs függvények tekintetében. A kialakított folyamatot erdészeti adatokon mutattam be, de a kialakítás során törekedtem az általánosságra, hogy az eljárás – minimális átalakítással – más adatokra, más területeken is használható legyen.

Következő lépés a Föld-rendszer projektben egy egységes táblázat létrehozása a független változókról, meteorológiai adatokról, ahol CReMIT segítségével az időben eltolt és időszakos aggregációk már elő vannak állítva. Ezekhez hozzáfűzve a függő változókat, legyen az bármilyen erdészeti adat, úgy végezhető adatelemzések, hogy nem- vagy csak minimális adatelőkészítési folyamatokra van szükség.

Köszönetnyilvánítás

Köszönetet mondok Jereb Lászlónak és Pödör Zoltánnak, akik bevontak 2013 júliusában a kutatási projektbe, lehetőséget adva ezzel a Kari Tudományos Diákköri Konferencián való részvételen és támogattak a munkámban, mikor szükség volt rá. A fejlesztések a 25294/207 TÁMOP-4.2.2.C-11/1/KONV-2012-0015 „Föld-rendszer adatok tudományos feldolgozása és az ismeretek társadalmasítása korszerű informatikai eszközök segítségével” című projekt kapcsán történtek.

Irodalomjegyzék

- [1] Pödör, Z., Edeléni, M., Jereb L. (2014): Systematic Analysis of Time Series. Infocommunication Journal, VI(1), p16-21.
- [2] R dokumentáció. <http://www.r-project.org/other-docs.html> (Utolsó megtekintés: 2014. április 9.)
- [3] R 'Test for outliers' csomag dokumentációja. <http://cran.r-project.org/web/packages/outliers/outliers.pdf> (Utolsó megtekintés: 2014. április 8.)
- [4] NIST/SEMATECH e-Handbook of Statistical Methods - 1.3.5.17.Detection of Outliers. <http://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm> (Utolsó megtekintés: 2014. április 8.)
- [5] Manninger M., Edeléni, M., Pödör, Z., Jereb, L. (2011): Alkalmazott elemzési módszerek a környezeti tényezők fák növekedésére gyakorolt hatásának vizsgálatában. Erdéstudományi Közlemények, 1(1), p59-70.
- [6] M. Ezekiel-K. A. Fox (1970) Korreláció- és regresszió-analízis – Lineáris és nem-lineáris módszerek. Közgazdasági és Jogi Könyvkiadó. Budapest. *A korrelációs mérőszámok jellegzetességei.* p157
- [7] Hajdu Ottó (2003) Többváltozós statisztikai számítások. Központi Statisztikai Hivatal. Budapest. *A multikollinearitás.* p247
- [8] Hajdu Ottó (2003) Többváltozós statisztikai számítások. Központi Statisztikai Hivatal. Budapest. *Stepwise algoritmusok.* p249
- [9] Edeléni Márton, Pödör Zoltán, Jereb László. Transzformált adatsorok alkalmazása a fák növekedése és az időjárási paraméterek kapcsolatának vizsgálatában, Agrárinformatika, Vol. II. 2011, No. 1:39-48, ISSN 2061-862X, p40-41.
- [10] Rapid-I GmbH (2012) How to Extend RapidMiner 5 White Paper. <http://1xltkxylmzx3z8gd647akcdvov.wpengine.netdna-cdn.com/wp-content/uploads/2013/10/How-to-Extend-RapidMiner-5.pdf> (Utolsó megtekintés: 2014. április 9)
- [11] RapidMiner 5.3.013 API dokumentáció. <http://fossies.org/dox/rapidminer-5.3.013/> (Utolsó megtekintés: 2014. március 25.)