Statisztikai Szemle

Közzététel: 2021. február 18.

A tanulmány címe:

**A gépi tanulás módszereinek alkalmazása R-ben**

Szerző:

**NYITRAI TAMÁS**, a Budapesti Corvinus Egyetem egyetemi adjunktusa
E-mail: tamas.nyitrai@uni-corvinus.hu

DOI: https://doi.org/10.20311/stat2021.2.hu0173

**Nyitrai Tamás**

# A gépi tanulás módszereinek alkalmazása R-ben

## Application of machine learning methods in R

NYITRAI TAMÁS, a Budapesti Corvinus Egyetem egyetemi adjunktusa
E-mail: tamas.nyitrai@uni-corvinus.hu

A technológiai fejlődésnek köszönhetően napjainkra a személyi számítógépek lehetővé tették a jellemzően nagy számításigényű gépi tanulásra épülő módszerek alkalmazását. Ezzel párhuzamosan az utóbbi években jelentős előrelépés történt a szabad hozzáférésű statisztikai, illetve adatelemző szoftverek funkcionalitásában is. A kedvező folyamatok ellenére a gépi tanulásra építő tanulmányok aránya jelentősen elmarad a hazai szakirodalomban a nemzetközihez képest. Ennek egyik lehetséges oka az, hogy ezen eljáráson alapuló modellezés magasabb szintű programozási ismereteket igényel. A tanulmány e nehézség elhárítását tűzi ki célul az R programnyelv egyik azon csomagjának bemutatásával, amely lehetőséget nyújt a gépi tanulásra épülő modellfejlesztésre, akár programozási ismeretek nélkül is.

TÁRGYSZÓ: gépi tanulás, R programozás, véletlen erdő

Due to technological advances, personal computers have made the application of machine learning methods characterised by high computational requirements, possible. In line with this, significant progress has been made in the functionality of open source statistical and data analysis software. In spite of these favourable processes, the number of Hungarian studies using machine learning methods lags behind that of articles published in the international literature. One possible reason for this may be that machine learning-based modelling requires a high level of programming skills. The aim of this article is to overcome this difficulty by presenting such an R package that allows model development using machine learning, without prior programming knowledge.

KEYWORD: machine learning, R programming, random forest

A tanulmány elsősorban technikai és módszertani jellegű: célja, hogy az Olvasó figyelmébe ajánlja a szabad hozzáférésű R szoftver egyik közelmúltbeli fejlesztését, amely a gépi tanulási módszerek alkalmazását jelentősen megkönnyíti. Az empirikus példánkban használt R csomag egyik legfontosabb sajátossága, hogy a

gépi tanulásra épülő modellépítés manuális programozást igénylő fázisainak automatizálását teszi lehetővé.

A témaválasztást a hazai szakirodalomban is megmutatkozó kedvező tendenciák motiválták, ugyanis a szabad hozzáférésű, statisztikai elemzésre is alkalmas programozási nyelvek és szoftverek használata egyre gyakoribbá vált az utóbbi években a magyar nyelven megjelenő publikációkban. Néhány önkényesen kiragadott példa a közelmúltból:

> – a Python programozási nyelv statisztikai alkalmazásához nyújt bevezető ismertetőt *Sóti* [2020];
>
> – *Abaligeti–Gyimesi–Kehl* [2020] az R programnyelv szabadon hozzáférhető adatforrások elérését lehetővé tevő módszereit mutatják be;
>
> – *Hajdu* [2018] tanulmányának célja „többváltozós statisztikai módszerekhez R kód szoftverfejlesztési R package felhasználási javaslatokat adni statisztikai módszerspecifikus alkalmazásokra" (*Hajdu* [2018] 1021. old.);
>
> – az R programnyelv főbb jellemzőit és népszerű csomagjait *Daróczi* [2016] munkája ismerteti;
>
> – *Vit* [2018] az előfordulási gyakoriságok modellezésekor alkalmazott (Poisson- és negatív binomiális, zéróinflált Poisson- és zéróinflált negatív binomiális, hurdle Poisson- és hurdle negatív binomiális) modelleket veti össze egy adatfelvétel keretében megkérdezett személyek roma ismerőseinek számát elemezve. A számításokat az idézett szerző szintén az R szoftverrel végezte.

Fontos kiemelni, hogy az idézett művek kiválasztása önkényes, azaz nem teljes körű irodalomfeldolgozás eredménye. Ennek ellenére véleményünk szerint jelzésértékű arra vonatkozóan, hogy a hazai szakirodalomban is egyre inkább megkerülhetetlenné válnak az olyan szabad hozzáférésű programnyelvek – illetve az ezekben írt statisztikai adatelemzés céljára készült szoftvercsomagok – mint az R vagy a Python.

Az említett programnyelvek mellett szintén egyre gyakrabban lehet találkozni a gépi tanulás (machine learning) módszereivel a hazai (a korábban idézett művek mellett lásd például *Futó* [2018] munkáját) és a nemzetközi szakirodalomban (például *Tripathi et al*. [2020]). Azonban ezen eljárások alkalmazásának gyakorisága – a hazai publikációkat tekintve – elmarad a „klasszikus" többváltozós statisztikai modellekétől. Néhány – szintén önkényesen kiragadott – példa a magyar nyelvű szakirodalomból a gépi tanulásra vonatkozóan:

> – *Ritzlné Kazimir–Máténé Bella* [2020] a $k$ legközelebbi szomszéd ($k$ nearest neighbour) eljárást választották a be nem fizetett általá-

nos forgalmi adó szintjének becslésére a hazai áfaalany nem pénzügyi vállalatok és egyéni vállalkozók körében;

– *Kristóf* [2018] hazai mikrovállalkozások csődelőrejelzésére az esetalapú következtetés (case-based reasoning) módszerét alkalmazta;

– *Uliha* [2015] az olajárak rövid távú előrejelzésére a neurális hálókat és az ún. tartóvektorgép (support vector machine) metódust használta.

Amint arra *Uliha* [2015] is rámutatott, a gépi tanulásban jelentős potenciál van a modellek illeszkedésének jóságát, illetve előrejelző képességét illetően. Ennek ellenére a közgazdasági és társadalomtudományok területén a hazai szakirodalomban viszonylag ritkán alkalmazzák a klasszikus többváltozós statisztikai módszerekhez képest. Jelen tanulmánynak nem célja e tendencia okainak pontos meghatározása, viszont nagy valószínűséggel szerepet játszhat benne az, hogy a gépi tanulás nagy számításigényű, illetve eddig haladó szintű programozási ismereteket követelt az elemzőktől. Utóbbi nehézség enyhítésére megjelentek ugyan grafikus felhasználói felületet kínáló szoftverek (például a Rapidminer), azonban ezek jelentős költsége szintén hátráltathatta a gépi tanulásra épülő eljárások elterjedését.

Napjainkra az R programnyelv számos olyan csomagja vált szabadon hozzáférhetővé, amelyek lehetővé teszik, hogy a gépi tanulási módszerek működési elveinek ismeretében bárki képes legyen azok használatára. Ráadásul mindez ma már mélyebb programozási ismeretek meglétéhez sem társul. Egy konkrét példán keresztül kívánjuk bemutatni az Olvasónak a gépi tanulásra épülő modellépítés folyamatát az adatok beolvasásától a felállított modell teljesítményének értékeléséig. A tárgyalás során a tevékenység egyes lépéseit megvalósító R utasításokat is felvonultatjuk, így az Olvasó akár ki is próbálhatja azokat.

A hitelkockázat kezelésében fontos szerepet játszó „rossz adósok” azonosításának klasszifikációs feladata kapcsán ismertetjük a gépi tanulási modellek felállításának folyamatát a véletlenerdő-módszer (random forest) példáján.

Tanulmányunk nem feltételez előismereteket sem a gépi tanulás, sem pedig az R programnyelv használatával kapcsolatban, de épít az Olvasó általános kvantitatív érdeklődésére és a valószínűségszámítás, illetve a többváltozós statisztikai számítások alapfogalmainak, főbb eljárásainak ismeretére.

## 1. Klasszifikációs feladatok megoldása gépi tanulásra épülő modellekkel

A gépi tanulásra épülő módszerek alapvető sajátossága, hogy nem élnek előzetes feltevésekkel a modellezni kívánt jelenséget leíró változók eloszlására, illetve a

függő és független változók kapcsolatának jellegére vonatkozóan, hanem azt az adatok alapján próbálják meg feltárni. A gépi tanulásra épülő modellek illeszkedésének mértékét döntően befolyásolja a modellek ún. hiperparamétereinek (beállításainak) igazítása a modellezni kívánt jelenség változóinak sajátosságaihoz. A hiperparaméterek optimalizációjával jelentősen javítható a modellek teljesítménye. E folyamat legfontosabb lépéseit mutatja be az 1. fejezet.

## 1.1. A hiperparaméterek optimalizációja

A hiperparaméterek optimalizációjából eredő teljesítményjavulás olyan mértékű is lehet, hogy az eljárás segítségével a modellépítéshez használt adatokon akár hiba nélküli eredmény is elérhető. Ezt a jelenséget nevezik a szakirodalomban túltanulásnak (*Hayashi–Oishi* [2018]), ami jelentős kockázatot jelent, mivel az empirikus adatok tartalmazhatnak hibás vagy egyéb okból extrém adatokat (zajokat), amelyek „megtanulása” nem lehet célja a modellfejlesztésnek. A túltanulás elkerülése érdekében a hiperparaméterek optimális értékének meghatározása során érdemes figyelembe venni, hogy a különböző paraméterkombinációk mellett miként alakul a modell teljesítménye olyan megfigyelések kapcsán, amelyek nem szerepeltek a modellépítéshez használt adatok között.

## 1.2. Modellvalidáció

A túltanulás elkerülése érdekében a rendelkezésre álló megfigyeléseknek csak egy részét használják fel a modell felállításához, majd annak előrejelző képességét a kihagyott mintaelemek vonatkozásában vizsgálják. Előbbit tanulási (*Kristóf* [2018]), utóbbit validációs mintaként ismeri a szakirodalom (*Giussani* [2019]). A validációs mintán elért eredményt azonban nagyban befolyásolhatja, hogy a rendelkezésre álló megfigyeléseket miként osztja fel a modell készítője tanulási és validációs mintákra, mivel egy adott validációs adathalmaz véletlenszerű kijelöléséből adódóan az ott kapott eredmény nem feltétlenül általánosítható a minta egészére. A probléma elkerülése érdekében a rendelkezésre álló adathalmazt általában többször osztják fel az elemzők véletlenszerűen tanulási és validációs mintákra, majd a vizsgált hiperparaméter-kombináció mellett a különböző validációs mintákon kapott eredmények átlagát tekintik az adott kombinációval felállított modell teljesítményére vonatkozóan objektív becslésnek.

A nemzetközi szakirodalomban általánosan elterjedt a többszörös keresztvalidáció módszerének alkalmazása (lásd például *Zhang et al.* [2020a] munkáját). Ennek lényege, hogy a megfigyeléseket *n* darab egyenlő részre osztjuk, ame-

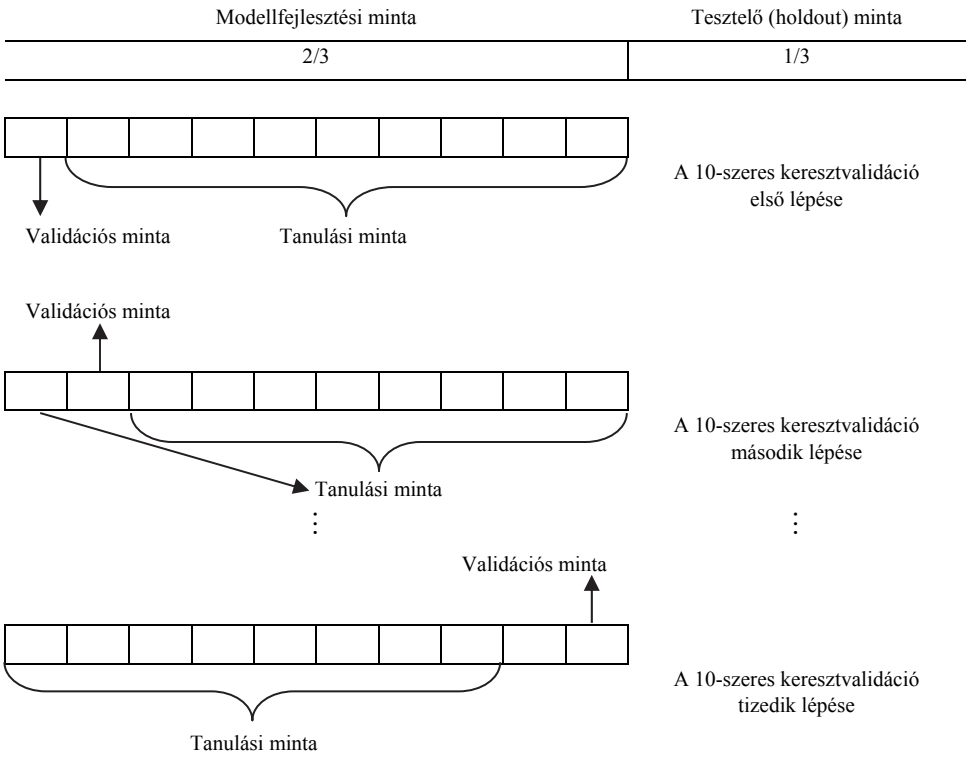
lyek közül egyet validációs mintaként, a fennmaradó $n - 1$ részt pedig együttesen tanulási mintaként használjuk fel. Az eljárás során minden rész szerepel egyszer validációs mintaként, azaz a folyamat $n$ darab iterációból áll, melynek mindegyik lépésében egy adott hiperparaméter-kombinációval felállított modell teljesítményét vizsgáljuk az épp kihagyott rész mint validációs minta adatain. Az $n$ darab validációs mintán kapott eredményt tekinthetjük a vizsgált hiperparaméter-kombinációval felállított modell teljesítményére vonatkozó becslésnek. Az $n$ értékét a rendelkezésre álló megfigyelések számosságának függvényében az elemző határozza meg, de nagyobb – 1 000 feletti – elemszám esetén jellemzően az $n = 10$ beállítást választják (lásd például *Kim* [2018] munkáját).

A többszörös keresztvalidációs eljárás megfelelő arra, hogy visszamérjük a modell teljesítményét különböző hiperparaméter-beállítások mellett. Arra a kérdésre azonban nem kapunk választ, hogy milyen teljesítményre számíthatunk, ha a legjobbnak vélt hiperparaméter-kombinációval felállított modellt olyan adatokon alkalmazzuk, amelyek nem szerepeltek az optimalizációs folyamatban.

E szempont figyelembevételét teszi lehetővé az angol terminológiával „holdout" módszernek nevezett technika, melynek lényege, hogy a megfigyelések egy részét „félretesszük" úgy, hogy azok semmilyen módon ne jelenjenek meg a modellfejlesztés folyamatában, és az adatállomány maradék részén a korábban bemutatott többszörös keresztvalidációt valósítjuk meg a legjobb teljesítményt nyújtó paraméterkombináció azonosítása céljából. Ezt követően az előzők alapján kiválasztott paraméterkombinációval felállított modell félretett részen végzett vizsgálatával azt tesztelhetjük, hogy milyen teljesítményre számíthatunk olyan adatok esetén, amelyek nem szerepeltek a modellépítéshez felhasználtak között. A fejlesztési és félretett részek arányát az elemző határozza meg a rendelkezésre álló megfigyelések számának függvényében. Ez a választás a félretett rész tekintetében gyakran esik az egyharmad arányra (lásd például *Sariev–Germano* [2020] munkáját).

Jelen tanulmányban is ezt a gyakorlatot követjük. A rendelkezésre álló megfigyelések kétharmad részét tekintjük modellfejlesztési mintaként, és azon belül alkalmazzuk a korábban bemutatott 10-szeres keresztvalidációt, melynek célja a legjobb klasszifikációs teljesítményt nyújtó hiperparaméter-kombináció (beállítás) azonosítása. Ezt követően a legmagasabb diszkrimináló erőt mutató paraméterekkel a modellfejlesztési mintában szereplő összes megfigyelést (az adatok kétharmad része) felhasználva felállítjuk a modellt, melynek teljesítményét a megfigyelések félretett egyharmad részén mint tesztelő mintán vizsgáljuk. A modellfejlesztési folyamatot az 1. ábra foglalja össze.

*1. ábra. A modellfejlesztési folyamat*
(Modell development process)




## 1.3. A klasszifikációs modellek főbb teljesítménymutatói

A statisztikai modellezés egyik fontos célja a függő változó minél pontosabb becslése. A valóságban megfigyelt és a modell által becsült értékek közötti eltéréseket összefoglaló jelleggel a modellek teljesítménymutatói számszerűsítik.

### 1.3.1. Találati arány

Klasszifikációs modellek esetén az egyik legkézenfekvőbb teljesítménymutató a találati arány, amely a helyesen besorolt megfigyelések számát viszonyítja a vizsgált megfigyelések teljes számához. Ennek számításához elegendő annyit ismerni, hogy a modell mely csoportba sorolta az egyes megfigyeléseket. Sok esetben felmerül a modellek eredményváltozójával szemben annak igénye, hogy azt is kifejezze, mennyire „határozott” a modell előrejelzése a megfigyelések egyes kategóriákba

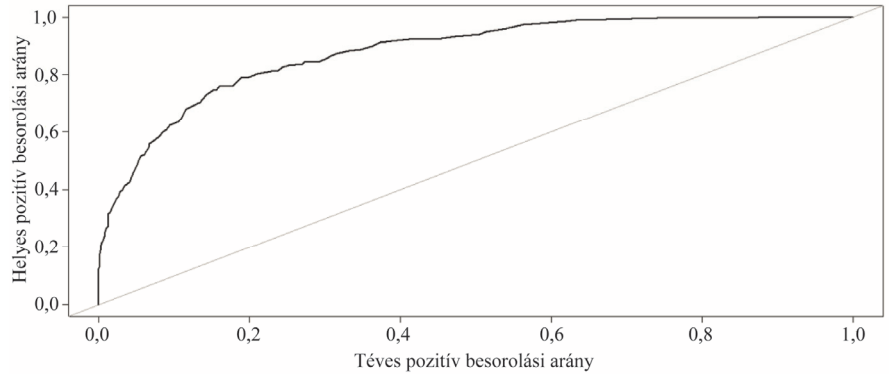
történő besorolása kapcsán. E számszerű érték azt fejezi ki, hogy a modell eredménye alapján mennyire valószínű, hogy a megfigyelt változók adott értékei mellett a kérdéses megfigyelés egyik vagy másik csoportba tartozik-e. Fontos kiemelni ugyanakkor, hogy a csoportképzéshez szükséges egy határértéket (cut value) választani, amely fölött (vagy alatt) az adott megfigyelést az egyik (vagy a másik) csoportba soroljuk (*Virág et al.* [2013]).

### 1.3.2. ROC-görbe

Szintén az elemző döntésétől függ az említett választóvonal meghatározása, amellyel az befolyásolható, hogy a modell milyen arányban sorolja be az egyes megfigyeléseket az elkülönített kategóriákba. A modellezni kívánt esemény bekövetkezését a nemzetközi szakirodalom jellemzően „pozitív" eseményként jelöli (lásd például *Zhang et al.* [2020b] munkáját). Ebből kiindulva a választóvonal módosítása azt is érinti, hogy milyen arányban sorolódnak be helyesen vagy tévesen az egyes megfigyelések a pozitív csoportba. Ezeket az arányokat nevezik helyes, illetve téves pozitív arányoknak (*Zhang et al*. [2020b]).

A kétértékű (bináris) klasszifikációs modellek teljesítményét gyakran vizsgálják az ún. kumulált besorolási pontosság (receiver operating characteristic, ROC) görbével, amely azt szemlélteti, hogy miként alakul az előbb említett arányok nagysága a két csoport elválasztására szolgáló választóvonal összes lehetséges értékét vizsgálva. A helyes pozitív besorolási arányt a függőleges, míg a tévest a vízszintes tengelyen ábrázoljuk. A gyakorlati alkalmazás során többnyire a pozitív megfigyelések minél pontosabb azonosítása a cél. Ilyenkor azon modellek preferáltak, amelyek ROC-görbéje minél inkább eltávolodik a vízszintes tengelytől. Ezt az eltávolodást méri számszerűen a ROC-görbe és a vízszintes tengely által határolt terület. Minél nagyobb ez a terület, annál jobb a modell klasszifikációs képessége.

*2. ábra. A tanulmány empirikus példájában felállított modell ROC-görbéje*
(ROC curve of the model developed for the empirical example of the study)



## 2. Az adatok előkészítése R-ben

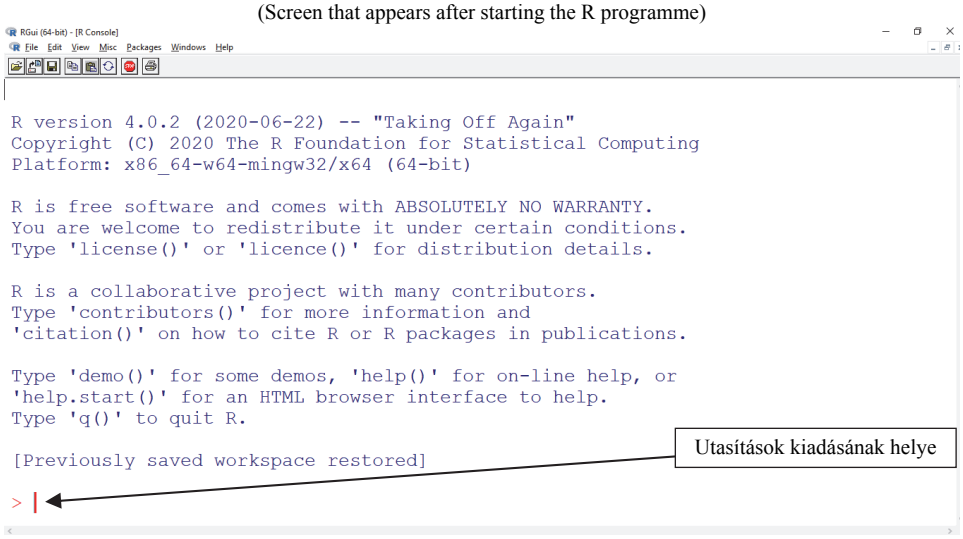
A fejezetben az **`mlr`** csomag telepítését, az adatok beolvasását, valamint modellezésre történő előkészítését ismerheti meg az Olvasó.

### 2.1. Az `mlr` nevű csomag telepítése

A tanulmány egyik fő célja, hogy azok is betekintést nyerjenek az R szoftverbe, akik korábban még nem használták, ezért a tárgyalás során mindössze annyit feltételezünk, hogy az Olvasó a számítógépére már telepítette a szabadon hozzáférhető R programot. Amennyiben erre még nem került sor, a https://www.r-project.org/ oldalon kiválasztható és telepíthető az R megfelelő verziója a felhasználó számítógépének, illetve az operációs rendszer típusának függvényében.

A program indítását követően megjelenik az ún. R konzol, ahol az Olvasó először egy rövid üdvözlő üzenetet lát. Az utolsó sorban az ábrán is látható > jelet követően van lehetőség a program számára utasításokat adni.

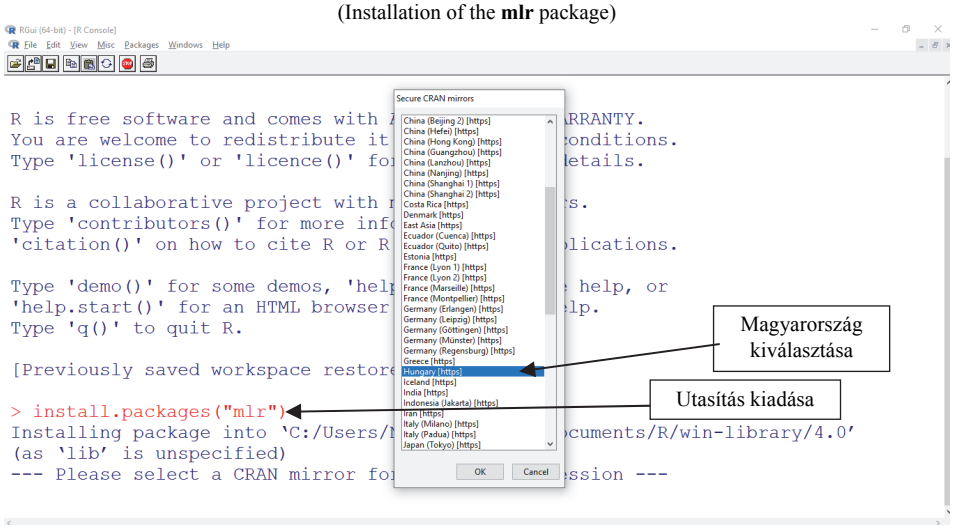
*3. ábra. Az R program indításakor megjelenő felület*
(Screen that appears after starting the R programme)




A program telepítését követően csak az alapvető funkcionalitás adott, melynek kifejtésére nem tér ki a tanulmány, azt részletesen *Daróczi* [2016], illetve *Hajdu* [2018] munkája tárgyalja. Az alapfunkciók azonban jelentősen bővíthetők a szintén szabadon hozzáférhető csomagok (packages) segítségével.

A gépi tanulásra épülő modellfejlesztés céljára számos csomag készült. Ezek közül az **`mlr`** csomag (*Bischl et al.* [2016]) használatába nyújtunk betekintést. Fontos kiemelni, hogy elemzésünk csak bevezető jellegű, terjedelmi okokból egyetlen tanulmány keretei között sincs lehetőség teljes körű áttekintésre, viszont a világhálón számos segédanyag[1] támogatja az Olvasót, ha mélyebben szeretné tanulmányozni az **`mlr`** csomagot. Az R szoftverhez a kiegészítő csomagok az ún. átfogó R archívum hálózaton (comprehensive R archive network, CRAN) keresztül érhetők el. A telepítés indítása után a rendszer kéri, hogy adjuk meg azt az országot, amelyen keresztül el kívánjuk érni az adott csomagot. Itt célszerű Magyarországot kiválasztani a legördülő menüből. Ezt követően lezajlik a telepítés.

*4. ábra. Az* **`mlr`** *csomag telepítése*
(Installation of the **mlr** package)




Az utasításokat pontosan kell kiadni, mivel akár egyetlen karaktereltérés is elegendő ahhoz, hogy a program válasza valamilyen hibaüzenet legyen, vagy – ami talán gyakorlati szempontból még hátrányosabb – nem azt az utasítást hajtja vére, illetve nem abban a formában, ahogy szerettük volna.

A hibaüzenet sokféle[2] lehet az utasítás kiadása során elkövetett hiba típusától függően, így ezek részletes kifejtésétől terjedelmi okok miatt eltekintünk. Sokkal

[1] A további részletek iránt érdeklődő Olvasó figyelmébe elsősorban a csomaghoz készült internetes oldalt ajánljuk: https://mlr.mlr-org.com/

[2] Például abban az esetben, ha a telepíteni kívánt csomag nevében (**`mlr`**) az első karakter helyett tévedésből „n”-t írunk (**`install.packages("nlr")`**), akkor a „package ‘nlr’ is not available” hibaüzenetet kapjuk, ami arra utal, hogy „nlr” néven nem érhető el csomag. Valamint, ha azt a hibát követjük el, hogy nem tesszük idézőjelbe a csomag nevét (**`install.packages(mlr)`**), akkor az „error in install.packages(mlr): object 'mlr' not found” hibaüzenet érkezik.

nehezebben kezelhető a hiba, ha a kiadott utasítás megfelel a programnyelv szabályainak, azonban nem az általunk megvalósítani kívánt módosítások következnek be.[3] A legnagyobb kockázat a modellezés eredményessége szempontjából, hogy ilyenkor nem kapunk hibaüzenetet, és ez a modellezési folyamat eredménytelenségét okozhatja. Emiatt célszerű időnként ellenőrizni, hogy valóban megfelelő utasításokat adtunk-e, továbbá azok a módosítások történtek-e, amelyeket meg szerettünk volna valósítani, illetve célszerű időnként menteni a folyamatot abban az esetben, ha esetleg egy hiba azonosítását követően vissza kell térnünk a modellezési folyamat azon pontjához, amelyet még nem érintett a felfedett hiba.

Érdemes még szót ejteni az R program által adott visszajelzések néhány főbb típusáról[4] is (*Matloff* [2011]).

– Hibára nem utaló visszajelzések:

– ha semmilyen visszajelzés nem érkezik, akkor a kiadott utasítást hiba nélkül végrehajtja a program;

– részletes visszajelzés érkezik az utasítás végrehajtásának eredményéről, amely általában valamilyen számítási művelet (például statisztikai próba) eredményét és esetleg annak részleteit közli.

– Hibára utaló visszajelzések:

– hibaüzenet (error): ilyenkor az utasítást nem hajtja végre a program. Ennek számos oka lehet. Az ok függvényében különböző hibaüzeneteket kaphatunk, ezek részletes bemutatásától terjedelmi okok miatt eltekintünk.

– figyelmeztető üzenetek (warning): nem szükségszerűen jelentenek hibát, illetve nem jelentik azt, hogy az utasítás eredményeképp nem azok a módosítások történtek, amelyeket valóban szerettünk volna. A figyelmeztető üzenetek általában tájékoztató jellegűek, amelyek befolyásolhatják a kapott eredmények értékelését. Ilyen például az, ha esetleg adathiba miatt egy eljárás valamely megfigyeléseket figyelmen kívül hagyott a számítások során.

[3] Ilyen eset lehet például az, amikor valamilyen felhasználó által meghatározandó paraméter értékénél követünk el gépelési hibát. A 4.4. alfejezetben például a rendelkezésre álló megfigyelések halmazát 2:1 arányban kívánjuk majd felosztani, amelyet a **`split=2/3`** utasítás kiadásával tehetünk meg. Ha esetleg tévedésből 1/3-ot írunk, akkor 1:2 arányban történik meg a felosztás, amely jelentősen eltérne az eredeti szándékunktól. A tévedésre azonban hibaüzenet nem fogja felhívni a figyelmet.

[4] Az R szoftverhez bárki készíthet és publikálhat szabadon hozzáférhető csomagokat. Az azokban elérhető utasítások és függvények definiálásakor a készítőnek lehetősége van egyedi hibaüzenetek, illetve figyelmeztető üzenetek megfogalmazására. A lehetséges visszajelzések formája szinte korlátlan, ezért csak a visszajelzések főbb típusait mutatjuk be érintőlegesen.

Fontos kiemelni, hogy egy telepített csomag automatikusan nem válik futtathatóvá, azt külön aktiválni kell a **`library()`** parancs segítségével, ahol zárójelben a futtatni kívánt csomag nevét szükséges megadnunk. Az **`mlr`** csomag aktiválását a következő utasítás kiadásával tehetjük meg:

```
>library("mlr")
```

## 2.2. Adatbeolvasás

Az R szoftver lehetővé teszi olyan adatok beolvasását is, amelyeket nem tárolunk a számítógépen, azonban ismerjük azt az internetes hivatkozást, ahol hozzáférhetők. Annak érdekében, hogy az itt bemutatott folyamatot az Olvasónak is lehetősége legyen kipróbálni, a tanulmányban a Statlog[5] (német hiteladatok) nevű, nyilvánosan hozzáférhető adatállományt használjuk. Az adatok beolvasására számos módszer áll rendelkezésre az R szoftverben. Ebben a tanulmányban a **`read.table()`** utasítást[6] használjuk. Ha közvetlenül internetes forrásból szeretnénk az adatokat beolvasni, akkor a zárójelben idézőjelek közé helyezve szükséges megadnunk azt az internetes hivatkozást, ahol az adatállomány hozzáférhető.

A beolvasni kívánt adatállománynak célszerű nevet adni annak érdekében, hogy a későbbiekben hivatkozni tudjunk rá a további utasítások kiadásakor. Az empirikus példában szereplő adatállománynak az **`adatok`** nevet adjuk. A hozzárendelést egyenlőségjel[7] segítségével végezzük.[8]

```
adatok=read.table("https://archive.ics.uci.edu/ml/machine-learning-
databases/statlog/german/german.data")
```

Látható, hogy az utasítás kiadását követően az Enter billentyűt lenyomva nem érkezik visszajelzés a programtól. Ez arra utal, hogy az adatok beolvasása és a hozzárendelés sikeresen megtörtént.

A **`str()`** utasítás segítségével átfogó képet kaphatunk a beolvasott adatállomány szerkezetéről. A zárójelben a vizsgálni kívánt adathalmaz nevét szükséges

[5] https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

[6] Az adatbeolvasás peremfeltételeinek részletes ismertetése meghaladja a tanulmány kereteit, azonban annak részleteiről a **`?read.table`** utasítás R konzolba történő begépelését követően megnyíló internetes oldalon tájékozódhat az Olvasó. Bármely utasítás kapcsán részletes dokumentáció érhető el az interneten, ha az utasítást tartalmazó csomag aktiválása után a kérdéses utasítás neve elé kérdőjelet írunk, és megnyomjuk az Enter billentyűt az R konzolban.

[7] Az R programozás gyakorlatában a hozzárendeléseknél használatos a **`<-`** jelzés is.

[8] Az utasítás terjedelmi okokból két sorban szerepel. A szövegben levő utasítás egyszerű kijelölése, másolása és beillesztése esetén hibaüzenet adódik, mert a szoftver nem tudja értelmezni a sortörés e formáját. A hiba elkerülése érdekében az utasítást folyó szövegként kell begépelni az R konzolba. Ugyanígy szükséges eljárni a 4.2. alfejezetben a **`parameterek`**, illetve a 4.5. alfejezetben a **`valid`** nevű objektum definiálásakor is.

megadni. Mivel az előző lépésben az **adatok** elnevezést használtuk, így az **str(adatok)** utasítást kiadva a következő eredményt kapjuk.

```
> str(adatok)
'data.frame':   1000 obs. of  21 variables:
 $ V1 : chr  "A11" "A12" "A14" "A11" ...
 $ V2 : int  6 48 12 42 24 36 24 36 12 30 ...
 $ V3 : chr  "A34" "A32" "A34" "A32" ...
 $ V4 : chr  "A43" "A43" "A46" "A42" ...
 $ V5 : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
 $ V6 : chr  "A65" "A61" "A61" "A61" ...
 $ V7 : chr  "A75" "A73" "A74" "A74" ...
 $ V8 : int  4 2 2 2 3 2 3 2 2 4 ...
 $ V9 : chr  "A93" "A92" "A93" "A93" ...
 $ V10: chr  "A101" "A101" "A101" "A103" ...
 $ V11: int  4 2 3 4 4 4 4 2 4 2 ...
 $ V12: chr  "A121" "A121" "A121" "A122" ...
 $ V13: int  67 22 49 45 53 35 53 35 61 28 ...
 $ V14: chr  "A143" "A143" "A143" "A143" ...
 $ V15: chr  "A152" "A152" "A152" "A153" ...
 $ V16: int  2 1 1 1 2 1 1 1 1 2 ...
 $ V17: chr  "A173" "A173" "A172" "A173" ...
 $ V18: int  1 1 2 2 2 2 1 1 1 1 ...
 $ V19: chr  "A192" "A191" "A191" "A191" ...
 $ V20: chr  "A201" "A201" "A201" "A201" ...
 $ V21: int  1 2 1 1 2 1 1 1 1 2 ...
> 
```

Amint látható, az output arról tájékoztatja a felhasználót, hogy a beolvasott adatállomány 1 000 megfigyelésre vonatkozóan 21 változó kapcsán tartalmaz adatokat. Az egyes sorok elején **V** előtaggal a változók nevei láthatók, mellette az **int** rövidítés az egészértékű, a **chr** a karakterként tárolt változókat jelzi.

Az adathalmaz első 20 változója egy bank 700 „jó” és 300 „rossz” adósára tartalmaz[9] számszerű adatokat és minőségi ismérveket. A **V21** változó 1-es értéke jelöli a „jó”, 2-es értéke pedig a „rossz” adósokat. A csoporttagságot jelölő változó valójában tehát minőségi ismérv, akárcsak az adathalmaz többi szövegként tárolt változója. A **V4** változó például az egyes ügyfelek hitelcélját, a **V9** pedig családi állapotát tartalmazza. Az egyes változók pontos tartalmáról a 9. lábjegyzetben található internetes hivatkozás részletes leírást közöl.

## 2.3. Adat-előkészítés

Az R szoftverben a minőségi ismérvek kezelésére szolgálnak az ún. faktorváltozók. Az adathalmaz szövegként tárolt minőségi ismérveit, illetve a modellezni kívánt **V21** függő változót a **factor()** utasítással alakíthatjuk át faktorváltozókká.

[9] https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

Az utasítás formája egy egyenlethez hasonlít. A bal oldalon látható **adatok** néven beolvasott adathalmaz minőségi ismérveit a jobb oldalon látható formára alakítjuk. Az utasításokban a **$** jel szolgál az adathalmazon belül a módosítani kívánt változó egyedi jelölésére, a jobb oldalon pedig a **factor()** utasítás végzi el a faktor változóvá történő konvertálást.

```
 adatok$V1=factor(adatok$V1)
 adatok$V3=factor(adatok$V3)
 adatok$V4=factor(adatok$V4)
 adatok$V6=factor(adatok$V6)
 adatok$V7=factor(adatok$V7)
 adatok$V9=factor(adatok$V9)
adatok$V10=factor(adatok$V10)
adatok$V12=factor(adatok$V12)
adatok$V14=factor(adatok$V14)
adatok$V15=factor(adatok$V15)
adatok$V17=factor(adatok$V17)
adatok$V19=factor(adatok$V19)
adatok$V20=factor(adatok$V20)
adatok$V21=factor(adatok$V21)
```

## 3. A véletlenerdő-eljárás[10]

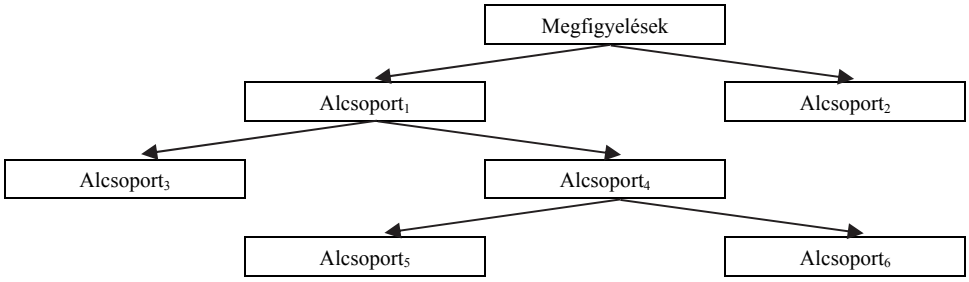
Az eljárás a döntési fákra épül, ezért célszerű először azok működésének alapelveit áttekinteni. A döntési fák a megfigyelt változók értékkészletét bontják részekre oly módon, hogy a létrejövő alcsoportok a modellezni kívánt függő változó tekintetében homogénebbek legyenek, mint a felosztás előtti állapotban. A felosztás az így létrejött alcsoportoknál is megtörténik mindaddig, amíg az alcsoportok homogenitásának növekedése még meghalad egy modellező által előzetesen beállított küszöbértéket. A bináris döntési fák a megfigyelések csoportját mindig két alcsoportra bontják (ilyenek például a klasszifikációs és regressziós fák), azonban léteznek olyan eljárások is, amelyek a megfigyeléseket kettőnél több alcsoportra bontják (ilyen például a khi-négyzet-alapú automatikus interakció-detektálás [chi-square automatic interaction detector, CHAID] módszere). A bináris döntési fák felépítését az 5. ábra szemlélteti.

A döntési fák általános sajátossága, hogy a felhasznált adatok kismértékű megváltoztatása jelentősen befolyásolhatja az eredményül kapott modellt, illetve annak teljesítményét. E hátrányos sajátosságot küszöböli ki az angol szaknyelvben „ensemble learningnek” (együttes tanulás) nevezett technika. Ennek lényege, hogy

[10] A módszer kidolgozása *Breiman* [2001] nevéhez köthető.

egy jelenség vizsgálatához az elemzők nemcsak egy modellt állítanak fel, hanem egyszerre többet, oly módon, hogy a rendelkezésre álló adatállománynak csupán egy véletlenszerűen kiválasztott részhalmazát használják fel, majd az így kapott modellek előrejelzéseit aggregálják.

*5. ábra. A bináris döntési fák felépítése*
(Structure of binary decision trees)




A több modell előrejelzésének aggregálásával kapott eredmény azonban csak akkor lehet pontosabb egy egyedi modellel kaphatóhoz képest, ha a véletlenszerűen kiválasztott részmintákon felállított modellek előrejelzései egymástól különböznek. A szakirodalom ezt nevezi az eredmények diverzitásának (*Xia et al*. [2020]), melynek forrása részben az, hogy az egyes modellek a megfigyelések teljes köréből vett véletlen mintákra épülnek, azonban ez önmagában gyakran nem biztosít elég diverzitást, és ezáltal jelentős javulást sem az egy modellel elérhető eredményhez képest. Az egyes modellekből kapott eredmények eltérése azonban növelhető azzal, ha nemcsak a megfigyelések köréből veszünk véletlen mintákat, hanem a független változók köréből is (*Tattar* [2018]).

Mindezek szemléltetésére tekintsünk egy olyan példát, ahol a vizsgálni kívánt jelenség megfigyelt egyedei (mintaelemei) egy táblázat soraiban, az azok kapcsán megfigyelt független változók értékei pedig annak oszlopaiban foglalnak helyet.

*6. ábra. A véletlen erdő működésének mintavételi sajátosságai*
(Sampling characteristics of the operation of random forest)

*p* darab változó kiválasztása

*n* darab megfigyelés kiválasztása visszatevéssel

| | Változó$_1$ | Változó$_2$ | Változó$_3$ | … | Változó$_k$ |
|---|---|---|---|---|---|
| Megfigyelés$_1$ | | | | | |
| Megfigyelés$_2$ | | | | | |
| Megfigyelés$_3$ | | | | | |
| ⋮ | | | | | |
| Megfigyelés$_n$ | | | | | |

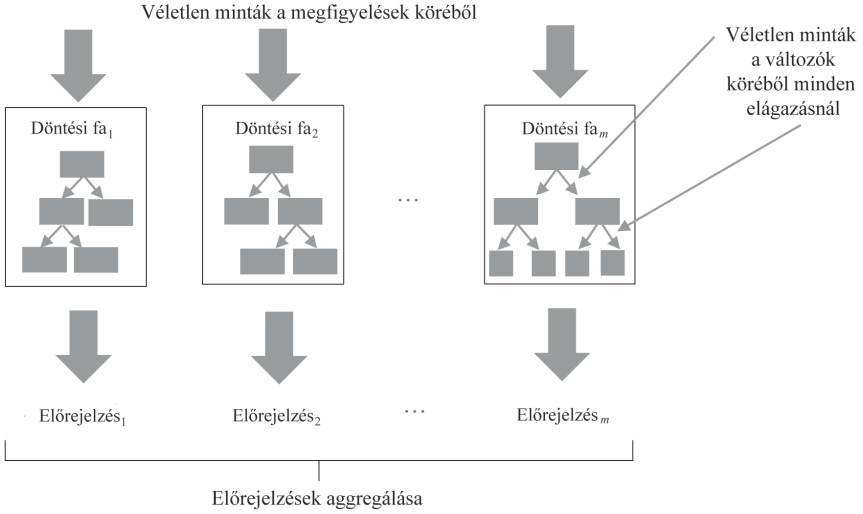
A 6. ábra egy olyan esetet mutat, ahol *n* darab megfigyelés kapcsán ismert *k* darab változó értéke. A véletlenerdő-módszer alkalmazása során több (*m* darab)

döntési fa jön létre. Az egyes fák felállításához az eljárás a rendelkezésre álló *n* darab megfigyelésből *n* elemű véletlen mintát vesz visszatevéssel, *m* alkalommal. Az így létrejött *m* darab véletlen minta lesz az *m* darab döntési fából álló véletlen erdő egyes tagjainak (döntési fáinak) modellfejlesztési adatállománya. Az első véletlen mintán a véletlen erdő első döntési fája, a második véletlen mintán a második döntési fája alapul, és végül hasonlóképpen, az *m*-edik véletlen minta megfigyelései képezik alapját a véletlen erdő *m*-edik döntési fájának.

Az így létrejövő döntési fák nem csak abban térnek el, hogy azok a rendelkezésre álló megfigyeléseknek csupán egy részhalmazán alapulnak. A modellépítéshez használt független változók köréből is mintavételre kerül sor, ami ebben az esetben a döntési fákon belül többször is ismétlődik. A véletlenerdő-módszer alkalmazásakor az egyes döntési fák minden elágazásánál újabb mintavétel történik a független változók köréből. Másképp megfogalmazva: a döntési fák egyes elágazásainál a rendelkezésre álló független változók közül csak $k < p$ darab visszatevés nélkül[11] kiválasztott változót veszünk figyelembe, ahol a *k* értékét a modell készítője határozza meg.

Mindezek alapján a véletlenerdő-módszer működését a 7. ábra szemlélteti.

*7. ábra. A véletlenerdő-módszer működése*
(Operation of the random forest method)




A 7. ábra bemutatja, hogy a véletlenerdő-módszer keretei között több döntési fa jön létre. E fák előrejelzéseinek aggregálásával határozható meg a véletlenerdő-

[11] A visszatevés nélküli mintavételi mód elágazásonként értendő (*Tattar* [2018]). Az eljárás minden elágazásnál kiválaszt visszatevés nélkül $k < p$ darab változót a döntési fa képzéséhez. Azonban a következő elágazásnál az előző lépésben kiválasztott *k* darab változó visszakerül, ezáltal a következő elágazás visszatevés nélküli mintavételében akár újra kiválasztható lesz.

modell előrejelzése. Utóbbi kapcsán azonban fontos újra kiemelni, hogy az egyes döntési fák az összes rendelkezésre álló megfigyelésből vett *n* elemű visszatevéses mintákra épülnek, azaz a megfigyelések egy – de döntési fánként változó – részét minden döntési fánál figyelmen kívül hagyjuk.

A véletlenerdő-módszer végső előrejelzésének meghatározása során csak azon döntési fák előrejelzéseit vesszük figyelembe, amelyek – véletlenszerűen kiválasztott – modellépítési mintájában nem szerepelt a vizsgálni kívánt megfigyelés (*Tattar* [2018]). Ezt szemlélteti a következő táblázat.

*A véletlenerdő-módszerrel felállított modell végső előrejelzésének meghatározása*
(Determining the final prediction of the random forest model)

| | $\text{Előrejelzés}_1$ | $\text{Előrejelzés}_2$ | $\text{Előrejelzés}_3$ | $\text{Előrejelzés}_4$ | $\text{Előrejelzés}_5$ |
|---|---|---|---|---|---|
| Megfigyelés | A | **B** | A | **B** | **A** |

A táblázat példájában csak 5 döntési fa szerepel a véletlenerdő-módszerrel felállított modellben egy olyan bináris klasszifikációs probléma megoldására, ahol az elkülöníteni kívánt csoportokat A, illetve B jelöli. Az 5 döntési fa közül 3 az A-val, 2 a B-vel jelölt csoportba sorolta a példában szereplő megfigyelést. Fontos kiemelni, hogy az 5 döntési fa közül csak a vastag betűvel kiemelt 3 fa esetén nem szerepelt a táblázat példájában szereplő megfigyelés az egyes fák alapját képező – véletlenszerűen kiválasztott – modellfejlesztési mintákban, ezért a véletlen erdő előrejelzésének meghatározásakor csak ezek előrejelzéseit vesszük figyelembe. E döntési fák közül a példában szereplő megfigyelést két modell (a 2-es és a 4-es) a B-vel, egy (az 5-ös) pedig az A-val jelölt csoportba sorolta. A véletlen erdők végső előrejelzését a többségi szavazás elvén határozzák meg a leggyakrabban (*Tattar* [2018]), azonban ettől eltérő módszer is alkalmazható. A példánk megfigyelését mindezek alapján a B-vel jelölt csoportba soroljuk a véletlenerdő-módszerrel.

## 4. A hiperparaméterek optimalizációja

A véletlenerdő-eljárásnak több hiperparamétere[12] van. Ezek közül terjedelmi okokból a bemutatandó optimalizációs folyamat csak a következő kettőre terjed ki:

– a döntési fák száma (neve: **`ntree`**);
– a fák egyes elágazásainál figyelembe vett, véletlenszerűen kiválasztott változók száma (neve: **`mtry`**).

[12] Az eljárás további paraméterei elérhetők a véletlenerdő-módszer R implementációjának módszertani dokumentációjában a **`randomForest`** csomag aktiválását követően a **`?randomForest`** utasítás segítségével.

## 4.1. A klasszifikációs feladat definiálása

R-ben történő modellezéskor célszerű minden fázist egy objektumban definiálni, amelyre a későbbiekben hivatkozni lehet az arra épülő további utasítások során. Első lépésként a klasszifikációs feladatot (**Task**) **feladat** néven definiáljuk a **makeClassifTask()** utasítással.

```
feladat=makeClassifTask(data=adatok,target="V21")
```

Az utasítás első paramétere a vizsgálni kívánt adatállomány neve – példánkban ez az **adatok** –, második paramétere pedig a felállítani kívánt modell függő változója. A következő lépés a használni kívánt klasszifikációs módszer definiálása **modszer** néven a **makeLearner()** utasítás segítségével.

```
modszer=makeLearner("classif.randomForest",predict.type="prob")
```

Az utasítás első paramétere a választott klasszifikációs eljárás neve, amelynek első tagja (**classif**) utal arra, hogy a módszert klasszifikációs célra kívánjuk használni. Az utasítás második tagja (**predict.type**) nem kötelező. Alkalmazása[13] lehetővé teszi, hogy az egyes megfigyeléseknél azt is mentse a rendszer, mennyi a pozitív (esetünkben a „rossz" adós) csoportba tartozás becsült valószínűsége a változók értékei alapján.

## 4.2. A paraméterhalmaz definiálása

A gépi tanulásra alapozott modellépítés legfontosabb lépése az eljárás beállításainak (hiperparamétereinek) optimalizálása. A feladatot nehezíti, hogy a paraméterek optimális értékeire sok esetben (különösen például az *Uliha* [2015] által használt tartóvektorgépek esetén) nincs objektív iránymutatás, így kísérletezéssel, illetve gyakorlati tapasztalatok felhasználásával történik meghatározásuk.

Példánkban az „erdőt alkotó" döntési fák számát (**ntree**) és az egyes elágazásoknál figyelembe vett, véletlenszerűen kiválasztott független változók számát (**mtry**) vizsgáljuk. A véletlen erdő implementációját tartalmazó **randomForest** nevű csomag alapbeállításai szerint az **ntree** értéke 500, az **mtry** értéke pedig a rendelkezésre álló független változók számának négyzetgyöke lefelé kerekítve.

A döntési fák számának növelésével jellemzően javul a véletlen erdő klasszifikációs képessége, ugyanakkor párhuzamosan növekszik a modell számításigénye.

[13] Elhagyása esetén a futtatást követően csak azt fogjuk látni, hogy a modell melyik csoportba sorolta a megfigyeléseket.

Mivel a vizsgálni kívánt adatállomány 1 000 megfigyelést tartalmaz, így talán kevésbé indokolt 500 döntési fa felállítása, ezért a döntési fák számát 100 és 400 között[14] vizsgáljuk az optimalizációs folyamatban. A döntési fák egyes elágazásainál a véletlenszerűen kiválasztott független változók számát az említett alapbeállítás (a 20 független változó számának négyzetgyöke) környezetében vizsgáljuk. Az optimalizációs folyamat az **mtry** mutató kapcsán a 4 és 5 értékeket tartalmazta.

A kijelölt 602 lehetséges paraméterkombináció vizsgálatának megvalósítása korábban mélyebb programozási ismereteket vagy rendkívül sok manuális munkát igényelt. Az **mlr** csomag ezeket a nehézségeket kiküszöböli, így a felhasználónak elég csak a paraméterhalmazt definiálni, és a keresztvalidációs tesztelést a csomag megfelelő utasításai elvégzik. Ehhez első lépésben a paraméterek körét szükséges meghatározni a **makeParamSet()** utasítással.

```
parameterek=makeParamSet(makeDiscreteParam("ntree",values=100:400),makeD
                  iscreteParam("mtry",values=4:5))
```

A **makeDiscreteParam()** parancs arra ad utasítást, hogy a **values** utáni, kettősponttal elválasztott[15] értékhatárok között minden egész értéket vegyen figyelembe a program. Az utasítás első paramétere az optimalizálandó paraméter neve idézőjelek közé helyezve.

Az optimalizáció során a 602 elemű paraméterhalmaz összes lehetséges kombinációját megvizsgáljuk. Ezt a **makeTuneControlGrid()** utasítással tehetjük meg, amelyet **tabl_opt** néven rögzítünk annak érdekében, hogy a további utasításokban hivatkozni tudjunk rá.

```
tabl_opt=makeTuneControlGrid()
```

## 4.3. A validációs módszer definiálása

A következő lépés a különböző paraméterkombinációkkal felállított modellek teljesítményének összevetésére szolgáló keresztvalidáció típusának definiálása. Erre a célra a **makeResampleDesc()** utasítás szolgál az **mlr** csomagban.

```
k_val=makeResampleDesc("CV",iters=10,stratify=TRUE)
```

A keresztvalidációnak a **k_val** nevet adjuk, és a későbbi utasításokban is így hivatkozunk rá. Az utasítás első paramétere (**"CV"**) a keresztvalidáció angol nyelvű

[14] A vizsgált intervallum kijelölése a szerző saját tapasztalatai alapján történt.

[15] Az R programban a kettőspont annak jelölésére szolgál, hogy két szélsőérték között – a végpontokat is beleértve – minden értéket egyes lépésközzel vizsgálunk. Például az 1:5 az 1, 2, 3, 4 és 5 értékek számsorát jelöli.

megfelelőjének rövidítése. Az **iters** paraméter a keresztvalidáció felosztásának darabszámát jelöli, amelyet korábban az 1.2. alfejezetben – a nemzetközi szakirodalom gyakorlatát követve (lásd például *Kim* [2018] munkáját) – 10-nek választottunk. A **stratify=TRUE** paraméter azt biztosítja, hogy a 10-szeres keresztvalidáció minden tizedében a „jó" és „rossz" adósok arányai azonosak legyenek a rendelkezésre álló teljes adathalmazra jellemző arányokkal.

## 4.4. A tesztelő minta kijelölése

A tesztelő minta kijelölésének elméleti vonatkozásait az 1.2. alfejezetben ismertettük. Meghatározása a gyakorlatban a 4.3. alfejezetben is használt **makeResampleDesc()** utasítással történik az **mlr** csomagban.

```
holdout=makeResampleDesc("Holdout",split=2/3,stratify=TRUE)
```

Az utasítás első paramétere a megfigyelések felosztásának típusa (ebben az esetben **"Holdout"**), a második (**split** nevű) paraméter rögzíti a modellfejlesztési adathalmaz arányát, a harmadik (**stratify=TRUE**) paraméter pedig most is azt biztosítja, hogy a tesztelő és modellfejlesztési adathalmazokban a „jó" és „rossz" adósok aránya azonos legyen a teljes adathalmazra jellemző arányokkal.

## 4.5. A tízszeres keresztvalidáció folyamatának definiálása

A következő lépés a keresztvalidáció folyamatának definiálása, amelyet a **makeTuneWrapper()** utasítással tehetünk meg.

```
valid=makeTuneWrapper(modszer,resampling=k_val,par.set=parameterek,contr
                      ol=tabl_opt,measures=list(auc,acc))
```

Az utasítás első paramétereként a klasszifikációs módszert szükséges megjelölnünk, amelynek korábban a **modszer** nevet adtuk. A **resampling** a modellfejlesztési adathalmazon alkalmazni kívánt validációs eljárásra utal, amelyet **k_val** néven határoztunk meg. A **par.set** a korábban **parameterek** néven rögzített paraméterhalmaz, amelynek összes elemét vizsgálni kívánjuk. Ezt korábban egy **tabl_opt** nevű objektumban definiáltuk, amelyet az utasítás **control** nevű paramétereként adtunk meg.

Az utasítás utolsó paramétere az optimalizációs folyamat célváltozójának meghatározása. Az empirikus példában választásunk – *Kristóf* [2018] munkáját követve –

a ROC-görbe alatti területre esett. A célváltozó megadására a **measures** paraméter szolgál. Az **mlr** csomag lehetőséget nyújt arra is, hogy egyszerre több teljesítménymutatót is megadjunk. Az optimalizáció az első mutató szerint történik, minden további mutató értékének tájékoztató jellegű rögzítése valósul meg.

A **measures** nevű paraméternél fontos kiemelni, hogy a teljesítménymutatókat lista (**list**) formában szükséges megadni. Esetünkben a lista első tagja az **auc**, amely a ROC-görbe alatti terület. Ennek tekintetében keressük a legmagasabb klaszszifikációs teljesítményt nyújtó paraméterkombinációt. A lista második elemeként megadott **acc** a vizsgált paraméterkombinációk mellett felállított modellek találati pontosságát jelöli.

## 4.6. Az optimalizációs folyamat végrehajtása

Az utolsó lépés az optimalizációs folyamat végrehajtása a **resample()** utasítás segítségével.

```
opt=resample(valid,task=feladat,resampling=holdout,measures=list(auc,acc))
```

Az utasításban az eddigiek során definiált objektumokat szükséges megadni:

– az első paraméter a korábban **valid** néven definiált keresztvalidációs eljárás, amelyet a modellfejlesztési adathalmazon kívánunk futtatni;

– a **task** nevű paraméter a klasszifikációs feladatot tartalmazza;

– a harmadik paraméter a **holdout**, amely meghatározza, hogy a rendelkezésre álló megfigyeléseket milyen arányban szeretnénk felosztani modellfejlesztési és tesztelő részekre. Továbbá ez a beállítás biztosítja azt is, hogy az első paraméternél kijelölt validációs eljárás alapján legjobbnak bizonyuló paraméterkombinációval felállított modell teljesítményét a félretett tesztelő („holdout”) adathalmaz megfigyelésein is megvizsgálja a szoftver;

– az utolsó paraméter a vizsgálni kívánt teljesítménymutatók listája.

## 4.7. Az optimalizációs folyamat eredménye, a felállított modell alkalmazása

A kézirat készítéséhez használt számítógépen[16] az optimalizációs folyamat mintegy 90 percet vett igénybe, melynek során a képernyőn megjelent a vizsgált 602 lehetséges paraméterkombináció és az azokkal felállított modellek klasszifikációs teljesítménye a 10-szeres keresztvalidáció átlagában. Terjedelmi okok miatt ezek közlésétől eltekintünk, és csak az optimalizációs folyamat outputjának utolsó, összefoglaló sorait mutatjuk be.

```
[Tune] Result: ntree=151; mtry=4 : auc.test.mean=0.7781383,
                    acc.test.mean=0.747748
          [Resample] iter 1:    0.8004060 0.7455090
```

Ezek az eredmények úgy értelmezhetők, hogy a 10-szeres keresztvalidáció átlagát tekintve abban az esetben mutatkozott a legjobb klasszifikációs teljesítmény a ROC-görbe alatti terület kapcsán, amikor a véletlenerdő-modellben 151 döntési fa szerepelt, és az egyes elágazásoknál véletlenszerűen kiválasztott független változók száma 4. E beállítások mellett a ROC-görbe alatti terület átlagosan 77,8, a találati arány pedig 74,8 százalék. A példánk szerinti paraméterkombináció alkalmazásával felállított modell a korábban tesztelési célból „félretett” – és a modellfejlesztési folyamat során nem használt – megfigyelések esetén 80 százalékos ROC-görbe alatti területet és mintegy 74,6 százalékos találati arányt ért el.

Fontos kiemelni, hogy a bemutatott modellezési folyamat megismétlése esetén az Olvasó a közöltektől eltérő eredményeket fog kapni. Ennek oka, hogy a programcsomag a keresztvalidáció, illetve a tanuló és tesztelő minták kijelölését minden esetben véletlenszerűen generált számok alapján hajtja végre. Emiatt az utasítások többszöri futtatása esetén az optimalizációs folyamatban nem lesznek teljesen azonosak a tanuló, validációs és tesztelő mintákban szereplő megfigyelések, következésképpen az azokra épülő modellek, illetve azok teljesítménye is különbözni fog valamelyest.

A bemutatott optimalizációs folyamat eredményeképp rendelkezésünkre áll a legjobb teljesítményt nyújtó kombináció (**`ntree`** = 151, **`mtry`** = 4). Ezek ismeretében lehetőség van a modellt a teljes rendelkezésre álló adathalmazra felállítani, továbbá a későbbiekben bármely olyan esetben alkalmazni, amikor az adatállományban szereplő független változók értékei alapján szeretnénk a „jó” és „rossz” adósokat klasszifikálni.

[16] Főbb jellemzők: 4 gigabájt RAM memória, Intel I3-4000M 2,4 GHz processzor, Windows 10 (64 bit) operációs rendszer.

Ehhez először a véletlen erdő (**"randomForest"**) csomagot szükséges telepíteni és aktiválni.

```
install.packages("randomForest")
    library("randomForest")
```

Majd sor kerül a modell felállítására a teljes adatállomány felhasználásával, a validációs folyamatban legjobbnak mutatkozó paraméterkombinációkkal. Ezt a véletlenerdő-módszer esetén a következő utasítással tehetjük meg.

```
modell=randomForest(V21~.,data=adatok,ntree=151,mtry=4)
```

A definíció bal oldalán a modell neve (**modell**) szerepel. A jobb oldalon először a választott módszer alkalmazását lehetővé tevő utasítást szükséges kiadni, ez a véletlen erdő esetén a **randomForest**. A zárójelben a függő és független változókat kell megadni ~ jellel elválasztva. Korábban a „jó" és „rossz" adósok csoportjait a **V21** változó tartalmazta, ezért ez kerül függő változóként a ~ jel bal oldalára. A jel jobb oldalán levő pont azt jelenti, hogy a függő változót a vizsgálni kívánt adatállomány összes többi változójával kívánjuk magyarázni. A **data** paraméternél a vizsgálni kívánt adatállomány nevét szükséges rögzíteni, ezt korábban **adatok** néven határoztuk meg. A további paraméterek az optimalizációs folyamat során legjobbnak mutatkozók értékei az előző alfejezetből.

A modell előrejelzéseit a **predict** nevű utasítással számíthatjuk ki.

```
pred=predict(modell,newdata=adatok)
```

Ez az utasítás az előrejelzéseket egy **pred** nevű objektumba menti. A **predict** parancs első paramétere a modell neve (példánkban **modell**), a **newdata** paraméternél pedig azon adathalmaz nevét (példánkban ez az egyszerűség kedvéért a modellépítéshez is felhasznált **adatok**) adjuk meg, amelynek megfigyelésein a modellt alkalmazni szeretnénk. A **newdata** elnevezés azt sugallja, hogy olyan adatállomány esetén is adhatunk előrejelzéseket, amelynek megfigyelései nem szerepeltek a modellépítéshez használtak között. Ennek azonban feltétele, hogy az itt megadott adatállomány ugyanazon változókat ugyanolyan formában[17] tartalmazza, mint a modell felállításához használt adatállomány.

[17] Azonos néven és változótípusokban tárolva.

## 4.8. A modell teljesítményének értékelése

Modellünk teljesítménye első látásra alacsonynak tűnhet az Olvasó számára. Ennek egyik oka, hogy célunk a gépi tanulásra épülő modellezés fontosabb lépéseinek bemutatása volt. A lényegi mondanivaló jobb kiemelése érdekében a hiperparaméterek optimalizációs folyamatát jelentősen egyszerűsítettük: csak két paraméter értékeit vizsgáltuk[18] egy viszonylag szűk intervallumban. A paraméterhalmaz definiálásakor azonban lehetőség van ezek, illetve az eljárás más paraméterei vonatkozásában is szélesebb intervallumok vizsgálatára. Ez azonban jelentősen növeli a lehetséges kombinációk számát és ezzel párhuzamosan az optimalizációs folyamat számításigényét, ugyanakkor valószínűleg lehetővé tenné a modell teljesítményének további javítását is.

A tanulmány példájának szabadon hozzáférhető adatállományát a nemzetközi szakirodalomban széles körben használják módszertani összehasonlító elemzések keretei között, ami lehetővé teszi az eredmények összevetését. A teljesség igénye nélkül egy önkényesen kiragadott példa *Boughaci et al.* [2020] munkája, ahol szintén 10-szeres arányos keresztvalidációt alkalmaztak ugyanezen az adatállományon a véletlenerdő-módszer vizsgálata során. A modellek átlagos találati aránya 76,4 százalék, ROC-görbe alatti területe pedig 79,1 százalék volt, amelyek nem haladják meg jelentősen a jelen tanulmányban közölt értékeket. Ugyanakkor az idézett szerzők kísérletet tettek a véletlenerdő-módszer kombinációjára a *k*-közép klaszterezési módszerrel, melynek köszönhetően jelentősen javult a véletlenerdő-módszerrel elérhető klasszifikációs teljesítmény. Mindez rámutat arra, hogy a példánkban használt német adatállomány változói alapján a „jó” és „rossz” adósok klasszifikációja további számottevő modellezési erőfeszítést (például módszerek kombinációja) igényel a magasabb előrejelző képesség érdekében. Ez azonban már túlmutat írásunk témáján és keretein.

# 5. Összegzés

A gépi tanulási módszerekkel történő klasszifikációs modellépítést mutattuk be az R programnyelv segítségével. A hitelkockázati adósminősítés példáján szemléltettük ennek folyamatát, melyhez az R `mlr` csomagját használtuk.

Kitértünk a gépi tanulásra épülő modellezés legfontosabb lépéseire, és rövid áttekintést adtunk a véletlenerdő-eljárás működési elvéről, illetve fontosabb para-

[18] A többi paraméterre vonatkozóan nem adtunk meg konkrét vagy vizsgálandó értékeket. Ilyen esetben az R azokat a szoftver alapbeállítása szerinti értékekre állítja be.

métereiről. Terjedelmi okokból ismertetésére csak a klasszifikációs feladatok példáján volt lehetőség, azonban fontos kiemelni, hogy a módszer alkalmazható regressziós, illetve klaszterezési problémák megoldására is (*Tattar* [2018]). Jelentős előnye a véletlenerdő-módszernek, hogy képes kezelni a hiányzó értékek problémáját, nincs szükség a kiugró értékek torzító hatásával kapcsolatos adat-előkészítő feladatok végrehajtására, illetve a multikollinearitásból eredő esetleges nehézségek kezelésére sem.

A kedvező sajátosságok mellett fontos felhívni a figyelmet az eljárás korlátaira is. Ezek egyike, hogy a véletlenerdő- és a legtöbb gépi tanulásra épülő módszer outputja általában nem értelmezhető. Az eredményül kapott modell nem ad olyan kézzel fogható „eszközt" az elemző vagy döntéshozó kezébe, mint például egy regressziós modell. A tanulmány példájában 151 döntési fa alkotta a véletlenerdő-módszerrel felállított modellt, melynek értelmezése ilyen nagy számosság mellett gyakorlatilag lehetetlen. Ugyanakkor az eljárás keretei között lehetőség van az egyes változók fontosságának mérésére, például annak vizsgálatával, hogy a véletlenerdő-módszerrel épített modellben hány döntési fában szerepeltek, illetve milyen mértékben befolyásolták a modell végső outputját. Ezek bemutatására sajnos terjedelmi okok miatt nem volt lehetőségünk.

Példánkban egy 1 000 elemű adatállományt vizsgáltunk 20 független változó tekintetében. A véletlenerdő-modell kapcsán 2 hiperparaméter optimalizálását mutattuk be részletesen, melynek mintájára az optimalizációs folyamat kiterjeszthető több hiperparaméterre, illetve szélesebb intervallumok vizsgálatára is. A 602 elemű paraméterhalmaz elemei közül a legjobb teljesítményt mutató kombináció meghatározása mintegy 90 percet igényelt a kézirat készítéséhez használt számítógépen. Nagyobb adathalmaz esetén és a vizsgálni kívánt paraméterek körének bővítésével párhuzamosan a számításigény exponenciálisan növekszik. Utóbbi korlát azonban a számítógépes hardverek (például többmagos processzorok alkalmazása), illetve a párhuzamos számítási kapacitások fejlődésének (*Wiley–Wiley* [2019]) köszönhetően egyre kevésbé akadályozza a gépi tanulásra épülő módszerek valós életből származó – jellemzően egyre nagyobb méretű – adathalmazokon történő használatát.

A bemutatott modell teljesítménye nem bizonyult kiemelkedőnek, azonban ennek maximalizálása nem volt célunk a modellezési folyamat egyszerűbb bemutatása érdekében. Fontos kiemelni, hogy a példánkhoz választott, nyilvánosan hozzáférhető adatállományt a nemzetközi szakirodalomban széles körben használják különböző modellezési koncepciók teljesítményének összevetésre. Ilyenek például *Tripathi et al.* [2020], illetve *Xia et al.* [2020] munkái. Az idézett művek referenciaként szolgálhatnak az érdeklődő Olvasó számára abban az esetben, ha meg szeretné ítélni saját modellezési koncepciójának klasszifikációs teljesítményét a német hiteladósok adatain.

Fontos hangsúlyozni, hogy csak a legfontosabb fogalmakat és modellezési koncepciókat mutattuk be a véletlenerdő-módszeren keresztül. Az **`mlr`** csomag funk-

cionalitása azonban ennél jóval tágabb. Ennek kapcsán az Olvasó figyelmébe a csomag készítői által létrehozott https://mlr-org.com/ oldal tanulmányozását ajánljuk, ahol elérhetők az **`mlr`** csomag további modellezési lehetőségei, és gyakorlati példák találhatók a klasszifikációs feladatok mellett a regressziós és klaszterezési alkalmazásokra is. A gépi tanulási módszerekre az R **`mlr`** csomagján kívül a CRAN-archívumban számos további csomag áll rendelkezésre. Választásunk azért esett az **`mlr`** csomagra, mert eljárások széles körét teszi elérhetővé egységes keretbe foglalva, jelentősen megkönnyítve a gépi tanulási módszerek használatát azok számára is, akik nem rendelkeznek mélyebb programozási ismeretekkel, de az alkalmazni kívánt eljárás elvi alapjait ismerik.

## Irodalom


ABALIGETI G. – GYIMESI A. – KEHL D. [2020]: Adatforrások használata R-ben. *Statisztikai Szemle.* 98. évf. 7. sz. 858–884. old. https://doi.org/10.20311/stat2020.7.hu0858

BISCHL, B. – LANG, M. – SCHIFFNER, J. – RICHTER, J. – STUDERUS, E. – CASALICCHIO, G. – JONES, Z. [2016]: „mlr: Machine learning in R". *Journal of Machine Learning Research.* Vol. 17. No. 170. pp. 1–5.

BREIMAN, L. [2001]: Random forests. *Machine Learning*. Vol. 45. October. pp. 5–32. https://doi.org/10.1023/A:1010933404324

BOUGHACI, D. – ALKHAWALDEH, A. A. K. – JABER, J. J. – HAMADNEH, N. [2020]: Classification with segmentation for credit scoring and bankruptcy prediction. *Empirical Economics.* 1 July. https://doi.org/10.1007/s00181-020-01901-8

DARÓCZI G. [2016]: Alkalmazott statisztika? R! *Statisztikai Szemle.* 94. évf. 11–12. sz. 1108–1122. old. https://doi.org/10.20311/stat2016.11-12.hu1108

FUTÓ I. [2018]: Mesterségesintelligencia-eszközök – logikai következtetésen alapuló szakértői rendszerek – alkalmazása a közigazgatásban, hazai lehetőségek. *Vezetéstudomány*. XLIX. évf. 7–8. sz. 40–51. old. https://doi.org/10.14267/VEZTUD.2018.07–08.05

GIUSSANI, A. [2019]: *Applied Machine Learning with Python*. Bocconi University Press. Milano.

HAJDU O. [2018]: Többváltozós statisztikai R Open alkalmazások. *Statisztikai Szemle.* 96. évf. 10. sz. 1021–1047. old. https://doi.org/10.20311/stat2018.10.hu1021

HAYASHI, Y. – OISHI, T. [2018]: High accuracy-priority rule extraction for reconciling accuracy and interpretability in credit scoring. *New Generation Computing.* Vol. 36. August. pp. 393–418. https://doi.org/10.1007/s00354-018-0043-5

KIM, S. Y. [2018]: Predicting hospitality financial distress with ensemble models: The case of US hotels, restaurants, and amusement and recreation. *Service Business.* Vol. 12. February. pp. 483–503. https://doi.org/10.1007/s11628-018-0365-x

KRISTÓF T. [2018]: A case-based reasoning alkalmazása a hazai mikrovállalkozások csődelőrejelzésére. *Statisztikai Szemle.* 96. évf. 11–12. sz. 1109–1128. old. http://doi.org/10.20311/stat2018.11-12.hu1109

KRISTÓF T. – VIRÁG M. [2019]: A csődelőrejelzés fejlődéstörténete Magyarországon. *Vezetéstudomány*. 50. évf. 12. sz. 62–73. old. http://doi.org/10.14267/ VEZ TUD.2019.12.06

MATLOFF, N. [2011]: *The Art of R Programming: A Tour of Statistical Software Design*. No Starch Press. San Francisco.

RITZLNÉ KAZIMIR I. – MÁTÉNÉ BELLA K. [2020]: A gazdasági és a szabályozási környezet változásának hatása az áfaelkerülés 2006 és 2016 közötti alakulására Magyarországon. *Statisztikai Szemle.* 98. évf. 2. sz. 107–132. old. https://doi.org/10.20311/stat2020.2.hu0107

SARIEV, E. – GERMANO, G. [2020]: Bayesian regularized artificial neural networks for the estimation of the probability of default. *Quantitative Finance.* Vol. 20. No. 2. pp. 311–328. https://doi.org/10.1080/14697688.2019.1633014

SÓTI A. [2020]: A Python programozási nyelvről statisztikusoknak. *Statisztikai Szemle.* 98. évf. 4. sz. 324–352. old. https://doi.org/10.20311/stat2020.4.hu0324

TATTAR, P. N. [2018]: *Hands-On Ensemble Learning with R.* Packt. Birmingham.

TRIPATHI, D. – EDLA, D. R. – KUPPILI, V. – DHARAVATH, R. [2020]: Binary BAT algorithm and RBFN based hybrid credit scoring model. *Multimedia Tools and Applications*. Vol. 79. 25 August. pp. 31889–31912. https://doi.org/10.1007/s11042-020-09538-6

ULIHA G. [2015]: Rövid távú olajár-előrejelzések teljesítményének stabilitása. *Statisztikai Szemle.* 93. évf. 3. sz. 189–224. old.

VIRÁG M. – KRISTÓF T. – FIÁTH A. – VARSÁNYI J. [2013]: *Pénzügyi elemzés, csődelőrejelzés, vállalati válságkezelés.* Kossuth Kiadó. Budapest.

VIT E. [2018]: A zéróinflált és a hurdle-modellek egy lehetséges társadalomtudományi alkalmazása: roma ismerősök számának elemzése. *Statisztikai Szemle.* 96. évf. 7. sz. 683–708. old. https://doi.org/10.20311/stat2018.07.hu0683

WILEY, M. – WILEY, J. F. [2019]: *Advanced R Statistical Programming and Data Models*. Apress. New York. https://doi.org/10.1007/978-1-4842-2872-2

XIA, Y. – ZHAO, J. – HE, L. – LI, Y. – NIU, M. [2020]: A novel tree-based heterogeneous ensemble method for credit scoring. *Expert Systems with Applications.* Vol. 159. No. 113615. https://doi.org/10.1016/j.eswa.2020.113615

ZHANG, Y. – LIU, R. – HEIDARI, A. A. – WANG, X. – CHEN, Y. – WANG, M. – CHEN, H. [2020]: Towards augmented kernel extreme learning models for bankruptcy prediction: Algorithmic behavior and comprehensive analysis. *Neurocomputing.* In press. Available online: 22 October. https://doi.org/10.1016/j.neucom.2020.10.038

ZHANG, X. – OUYANG, R. – LIU, D. – XU, L. [2020]: Determinants of corporate default risk in China: The role of financial constraints. *Economic Modelling.* Vol. 92. November. pp. 87–98. https://doi.org/10.1016/j.econmod.2020.07.005