

BME Matematika Intézet

# Automatizált változó generálás

SZAKDOLGOZAT

Kubicza Gréta Andrea

Témavezető: Dr. Kovács Edith Alice

egyetemi docens

BME Matematika Intézet

Differenciálegyenletek Tanszék



2021. május 21.



# Önállósági nyilatkozat

Alulírott Kubicza Gréta Andrea a Budapesti Műszaki és Gazdaságtudományi Egyetem matematika BSc szakos hallgatója kijelentem, hogy ezt a szakdolgozatot meg nem engedett segédeszközök nélkül, önállóan, a témavezető irányításával készítettem, és csak a megadott forrásokat használtam fel.

Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból vettettem, a forrás megadásával jelöltetem.

Budapest, 2021. május 21.

Kubicza Gréta Andrea

## Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani a témavezetőmnek, Kovács Edith Alicenek az érdekes témáért, a témában való segítségért, a szakdolgozat írásában nyújtott tanácsokért és segítségért.

Köszönnettél tartozom szüleimnek és testvéreimnek, a szeretetért és a támogatásért, amit tanulmányaim során kaptam, kapok.

Külön köszönöm páromnak, Mártonnak a sok türelmet, szeretetet és támogatást, mellyel hozzájárult a szakdolgozat elkészüléséhez.

## Tartalomjegyzék

<b>1. Bevezető</b>	<b>6</b>
<b>2. Feature engineering</b>	<b>7</b>
2.1. Hiányzó adatok, értékek kezelése . . . . .	8
2.2. Duplikáció kezelése . . . . .	8
2.3. Kilógó értékek (outliers) kezelése . . . . .	8
2.4. Diszkretizálás . . . . .	8
2.5. Logaritmikus transzformálás . . . . .	9
2.6. Skálázás . . . . .	9
2.7. Bináris átírás (one-hot encoding) . . . . .	9
2.8. Műveletek csoportosítása . . . . .	10
2.9. Változó bontás . . . . .	10
2.10. Dátum kezelése . . . . .	10
2.11. Új változók generálása . . . . .	10
<b>3. Feature expansion autofeat segítségével</b>	<b>11</b>
3.1. Változók generálása . . . . .	11
3.2. Változók kiválasztása . . . . .	13
3.3. Az autofeat regressziós és osztályozós függvényei . . . . .	14
3.3.1. Osztályozás . . . . .	15
3.3.2. Regresszió . . . . .	15
<b>4. Az autofeat eljárással kapcsolatos kérdések, feltevések</b>	<b>16</b>
<b>5. Az alkalmazott modellek és dimenziócsökkentő eljárások ismertetése</b>	<b>17</b>
5.1. Szupport vektor gépek (SVMs) . . . . .	19
5.1.1. Osztályozás (Classification) . . . . .	20
5.1.2. Regresszió (Regression) . . . . .	21
5.2. Lineáris Modellek . . . . .	22
5.2.1. Ridge regresszió . . . . .	22
5.2.2. Logisztikus regresszió . . . . .	23
5.3. Együttes (ensemble) módszer . . . . .	23
5.3.1. Véletlen erdők . . . . .	24
5.4. Random Forests Feature importance és Boruta algoritmus . . . . .	26
5.4.1. Random Forests Feature importance . . . . .	26
5.4.2. Boruta algoritmus . . . . .	27
<b>6. Eredmények</b>	<b>28</b>

Tartalomjegyzék

---

7. Összefoglalás 37

Irodalomjegyzék 38

---

## 1. Bevezető

A nemlineáris összefüggéseket megtanuló gépi tanulási modellek, mint például a neurális hálók, döntési fák, véletlen erdők vagy regressziós modellek általában nehezen megmagyarázhatók az alkalmazók számára, mivel nem látják a változók közötti transzformációt zárt formában. Ezért sok esetben hasznos, a változók nemlineáris összefüggéseit úgy felfedezni, hogy ezeket zárt alakban látjuk. Sokszor a felhasználó az, aki szakértője az adott adatnak és több értékes információra tud szert tenni ezeket meglátva.

A feature engineering egy alapvető, de munkaigényes része a gépi tanulás alkalmazásának. A legtöbb gépi tanulási modell teljesítménye nagyban függ a bemeneti változóktól. Ennek eredményeképpen az adattudósok sok energiát fektetnek az adatok előkészítésébe, traszformálásába. Olyan kompatibilis formára hozzák az adathalmazt, amely már megfelel az algoritmus követelményeinek, mindeközben olyan változókat alakítanak ki, amelyekkel több információt nyerhetnek ki, javítva ezzel a modell teljesítményén. Az eredeti változók nemlineáris transzformációinak köszönhetően új változókat lehet definiálni. Ezek bevezetése a feature engineering egy speciális irányzata, amit feature expansionnak, azaz változó kibővítésnek neveznek. Ezáltal arra törekednek, hogy kifejezőbb, jobban használhatóbb attribútumokat hozzanak létre.

A változó kibővítésre specializált eljárásokat három fő csoportba sorolnám. Az egyik a kernelek segítségével növeli a dimenziót, vagyis belevetíti a mintaelemeket egy magasabb dimenziós térbe, ahol várhatóan jobb tulajdonságokkal rendelkezik az adathalmaz, mint például a Support Vector Machine (SVM) esetén. A másik csoport a neurális hálók, ezek közül az autoencoder neurális hálót említeném meg, amelyben először kibővülnek a köztes layerek, majd redukálódnak. A harmadik csoport pedig a változók nemlineáris transzformációját használja fel új válto zók generálására. Az utolsó csoport az előzőekkel szemben felhasználóbarát, hisz betekintést nyújt ezekbe az új változókba. Hátránya, hogy nagyon felduzzasztja a magyarázóváltozók halmazát.

A dolgozatomban leírt kutatásokat a "The `autofeat` Python library for Automated Feature Engineering and Selection" [1] cikk és hozzáartozó új Python `autofeat` [2] csomag inspirálta, amely az utolsó csoportra dolgozott ki egy automatikus eljárást. A vizsgált eljárás a kiinduló változók bizonyos nemlineáris transzformációit is beépíti a változók közé, majd azokat tartja meg, amelyek relevánsak a célváltozókra. Ha ezeket lineáris, illetve logisztikus regresszióval kombinálva használjuk, jól értelmezhető és magyarázható modelleket kapunk.

A dolgozatom felépítése a következő. A második részben ismertetem a feature engineering legismertebb eljárásait, a harmadik részben bemutatom az `autofeat` csomag legfőbb elemeit, a negyedik fejezetben megfogalmazom, hogy milyen kérdésekre szeretnék választ kapni, az ötödik részben szeretném bemutatni az alkalmazott és felhasznált modelleket, eljárásokat, majd a hatodik fejezetben ismertetem a levonandó következtetéseket és eredményeket.

## 2. Feature engineering

Az adatokat, információkat adathalmazokba gyűjtjük, amelyeket leggyakrabban táblázatok formájában tárolunk. Ezen táblázatok sorai a rekordok, oszlopai az attribútumok vagy változók, amelyeknek az értékei az egyes rekordokat jellemzik.

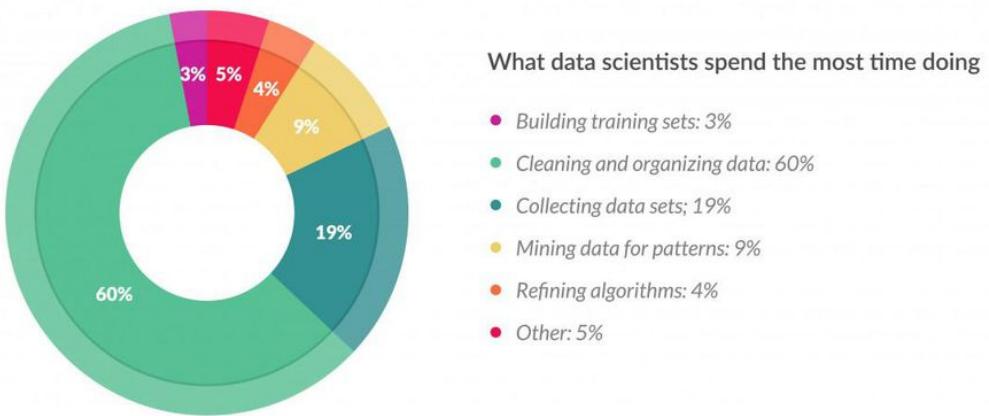
Alapvetően minden gépi tanulási algoritmus valamennyi bemeneti adatot használ fel, hogy meghatározza a kimeneti eredményt. Az algoritmusok megfelelő működéséhez szükség van a nyers adatok elfogadható, feldolgozható formára hozására, hogy a modellezésre alkalmas állapotba kerüljenek, ehhez van szükség a feature engineeringre.

**Feature engineering definíciója** [3]: Az a folyamat, amely során új attribútumokat készítünk a nyers adatból, hogy növeljük a tanuló algoritmusunk predikciós erejét. Az új attribútumokkal olyan további információkhoz juthatunk, amelyek látyszólag nem álltak rendelkezésünkre az eredeti attribútumhalmaz alapján.

A feature engineeringnek két legfontosabb célja [4]:

- A megfelelő bemeneti adathalmaz előkészítése, hogy kompatibilis legyen a gépi tanulási algoritmus követelményeivel.
- A gépi tanulási modellek teljesítményének javítása.

Egy felmérés szerint az adattudósok az idejük 80 százalékában az adatok előkészítésével foglalkoznak. Ebből is látszik, hogy mennyire fontos a feature engineering [5].



1. ábra. Diagramm az adatelőkészítés fontosságáról.

A következő alfejezetekben röviden ismertetve lesznek a feature engineering főbb technikái.

## 2.1 Hiányzó adatok, értékek kezelése

---

### Főbb feature engineering technikák

#### 2.1. Hiányzó adatok, értékek kezelése

A legtöbb esetben a legkönnyebb megoldás, az ha töröljük az adott sorokat vagy akár az oszlopot, amelyből hiányoznak az adatok. Egy másik megoldás az adatok pótlása, amelyet megtehetünk egy adott értékkel, például numerikus adatok esetén az átlaggal, a mediánnal vagy a módusszal, vagy kategorikus adatok esetén egy új kategória bevezetésével.

#### 2.2. Duplikáció kezelése

Az adathalmaztól függ, hogy mit szeretnénk kezdeni az adat duplikációval. Van olyan eset, amikor nem kell vele foglalkozni, vagy adatduplikációra számítunk, mert információt ad számunkra az adatról, az adatok eloszlásáról. De vannak olyan esetek, amikor el szeretnénk kerülni a duplikációt, például amikor az adatok egyedisége a cél, ebben az esetben a megkettőződött adatokat érdemes törölni.

#### 2.3. Kilógó értékek (outliers) kezelése

A kilógó értékek keletkezhetnek mérési hibából, zajból, vagy valamilyen ritka eseményből is, ezért a felderítésükben különbözőképpen érdemes eljárni. Azt is fontos megjegyezni, hogy mit szeretnénk kinyerni az adatokból. A legjobb módja a kilógó értékek detektálására az adatvizualizáció, minden más statisztikai módszer esetén nagyobb hibázási lehetőség áll fenn. Az outlierek kiszűrésére használt statisztikai módszerek kevésbé precízek, de másfelől nagyon gyorsak. Ilyen módszer például az adott érték tapasztalati szórástól való eltérése, átlagos abszolút eltérése, kovarianciája, korrelációja vagy a percentilisekhez való viszonyítása. Ezek segítségével megállapíthatjuk, hogy statisztikai értelemben kilógó értékről van-e szó. Ezután a probléma alapján kell eldöntenи, hogy mi a jobb számunkra, törölni az adatot vagy megtartani, esetleg módosítani. Eddig ez egyes változók kilógó értékeiről volt szó. Ezenkívül vannak másfajta outlierek is, amik több változó együttes ingadozássával kapcsolatosak, például amikor nem a "megszokott" összefüggés van a változók között, ám ezeknek a detektálása külön kutatási irányzat, ez nem tartozik a feature engineering eljárások közé.

#### 2.4. Diszkretizálás

A diszkretizálás célja, hogy a folytonos változókat diszkrétté alakítsunk, azaz minden értéket kategóriákba akarunk sorolni, amelyek lehetnek diszjunktak vagy átfedők (általában diszjunkt). A diszkretizáció fő motivációja a modell robusztusabbá tétele és a túltanulás megakadályozása, azonban ennek ára is van. minden alkalommal, amikor valamit diszkretizálunk, információt is áldozunk fel és az adatunk

## 2.5 Logaritmikus transzformálás

---

letisztultabbá válik. Így kevesebb értéket kell tárolnunk és bizonyos algoritmusok futási ideje nagyban lecsökken. A teljesítmény és a túltanulás közti egyensúly a kulcsa a diszkretizálásnak.

### 2.5. Logaritmikus transzformálás

Ez a leggyakrabban használt matematikai traszformáció a feature engineering során. Előnyei a következők:

- Segít kezelni a ferde adatokat és a transzformáció utáni eloszlás jobban megközelíti a nomálist.
- Normalizálja a nagyságrendbeli különbségeket.
- Emiatt csökkenti a kilógó értékek hatását és a modell robusztusabbá válik.

Fontos megjegyezni viszont, hogy csak pozitív értékeken lehet alkalmazni, a logaritmus függvény értelmezési tartománya miatt.

### 2.6. Skálázás

A legtöbb esetben a numerikus adathalmaznak nincs meghatározott tartománya és nagyon különbözők egymástól. Például nem várhatjuk el, hogy az életkor és a bevétel hasonló tartománnyal rendelkezzen, ezáltal nem tudjuk őket megfelelően összehasonlítani. Erre nyújt megoldást a skálázás.

A leggyakoribb skálázási módszerek:

- Átskálázás a  $[0, 1]$  intervallumba:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (1)$$

- Standardizálás:

$$x' = \frac{x - \bar{x}}{s_x} \quad (2)$$

- Normálás:

$$x' = \frac{x}{\|x\|} \quad (3)$$

### 2.7. Binárissá átírás (one-hot encoding)

Ennél a módszernél egy adott oszlop minden értékéhez külön változót készítünk, és az  $i$ . változó értéke pontosan akkor lesz 1, ha az adott sorban az eredeti változó értéke  $i$  volt, különben 0.

## 2.8. Műveletek csoportosítása

Vannak olyan adathalmazok, mint például a tranzakciókezelések, amelyeknél előfordulhatnak többszörösen ismétlődő sorok. Ilyen esetben az adatokat adott értékek szerint csoportosítjuk, majd minden értéket csak egy sor képvisel. A műveletek szerinti csoportosítás kulcsa, hogy eldöntsük milyen függvény szerint szeretnénk csoportosítani. Numerikus változók esetén lehet szó az átlagról vagy az összegről, de a kategorikus változók esetében ez sokkal bonyolultabb.

## 2.9. Változó bontás

Néha hasznosabbak tudnak lenni a változók, ha több különböző változóra bontjuk őket. Ilyen változó legtöbbször a dátum-idő, a cím, a név, stb. Azzal, hogy több változóra bontjuk őket, új felhasználási lehetőségekhez jutunk, például:

- Elérhetővé válik a gépi tanulási algoritmusunk számára, hogy összehasonítsa őket.
- Lehetőségünk nyílik a diszkretizálásukra és a csoportosításukra.
- Növelhetjük a modellünk teljesítményét az újonnan felfedezett információkkal.

## 2.10. Dátum kezelése

A dátum általában értékes információt nyújt a célváltozó meghatározásához, de a gépi tanuló algoritmus elhanyagolja vagy értelmetlennek tartja a használatát. Ez főként azért lehet, mert nem megfelelő a formátuma, így az algoritmus nehezen tudja értelmezni. Annak érdekében, hogy könnyebben, több információt nyerjen ki, érdemes a következő formátumra módosítani a dátumot vagy új változókat meghatározni belőle:

- Külön változókra célszerű bontani az évet, a hónapot és a napot.
- Meghatározni a jelenlegi dátum és az adott dátum közti különbséget és az előző pontban megemlített formátumba feljegyezni.
- Meghatározhatjuk ezen felül még, hogy a dátumunk a hét melyik napjára esik, hétköznap vagy hétfége, munkanap vagy sem, stb.

## 2.11. Új változók generálása

Az új változók generálása során, a már meglévő változókból valamilyen módon teljesen új változókat generálunk, hogy növeljük a modellünk teljesítményét. Erre sok különböző technika létezik, például összekombinálhatunk egy vagy több változót egy új változó létrehozásához, amely hasznosabb lehet a modell számára. A továbbiakban ezzel a témakörrel még részletesebben foglalkozunk.

---

### 3. Feature expansion autofeat segítségével

Új változók létrehozása nem azt jelenti, hogy új méréseket végzünk új szempontból, hanem azt, hogy a létező adatokat felhasználva transzformációk segítségével képezzük őket.

Ebben a fejezetben a Python egy viszonylag új (2019-ben létrehozott) könyvtárának, az `autofeat`-nek [1] a változó generálás (feature generation) és a változó szelektálás (feature selection) részeit, valamint a két beépített predikciós modelljét szeretném ismertetni.

Az `autofeat` könyvtár `AutoFeatRegressor` és `AutoFeatClassifier` modelljei automatikusan generálnak és szelektálnak további nemlineáris bemeneti változókat az eredeti adatok alapján és egy lineáris predikciós modellt képeznek ezekkel az új változókkal.

A modellek az ismert `scikit-learn`-höz hasonló felületűek [2]:

- `fit()` függvény illeszkedik a modell paramétereire
- `predict()` függvény predikciót ad a célváltozóra bemeneti adatok alapján
- `score()` függvény kiszámolja a modell "jóságát" (az  $R^2$  mutatót)
- `fit_transform()` és `transform()` függvények kibővítik az megadott adatokat az új generált és szelektált változókkal

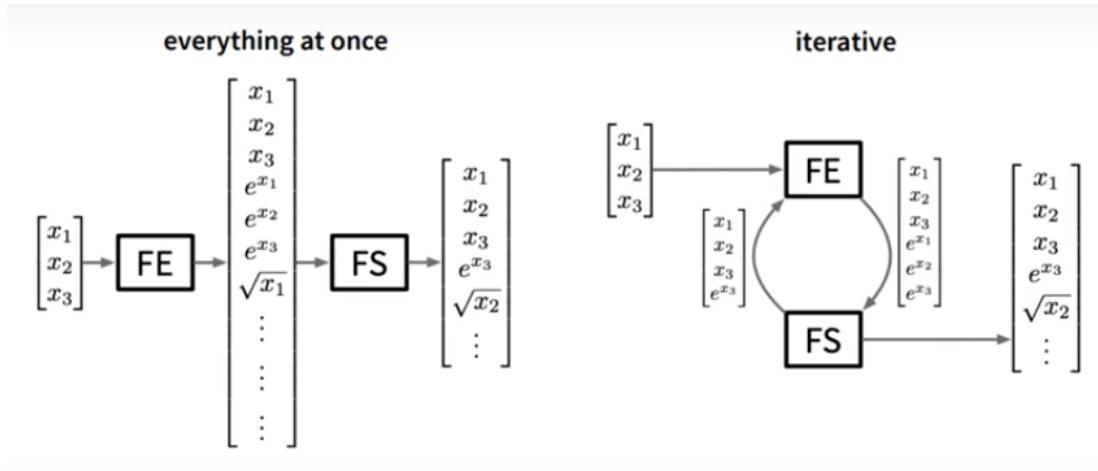
A következő alfejezetekben a változók generálásának és szelektálásának lépései szeretném ismertetni, amelyek az `AutoFeatRegressor.fit_transform()` vagy `AutoFeatRegressor.fit()` függvények hívása során történnek, valamint bemutatom magukat a modelleket is.

#### 3.1. Változók generálása

Az új változók generálása kétféle módon történhet, először legeneráljuk a változók egy nagy halmazát majd ebből választjuk ki a releváns változókat (ezt a stratégiát követi az `autofeat`) vagy iteratív módon generálunk egy kisebb új változó halmazt, ebből kiválasztjuk a relevánsakat, majd újra generálunk és szelektálunk, ezt ismételve addig, amíg javítani nem tudunk az új változókkal a predikció pontosságán.

Mindkettőnek megvan a maga hátránya, az első az nagyon memória igényes, főként amikor egy nagy bemeneti változóhalmazból kell konstruálni számos traszformációval az új változókat. A második esetben pedig fontos új változókat veszíthetünk el, ha túl korán töröljük a szelektációk során az alapjukként szolgáló változókat, ezáltal kevésbé tudunk komplex, lehetőleg hasznos változókat generálni. Továbbá a kiválasztott stratégiától függően az egész procedúra is igazán időigényes lehet, ha a modell minden lépésben rátanul és kiértékel a változók részhalmazára vagy releváns

### 3.1 Változók generálása



2. ábra. Változó generáló stratégiák.  
 FE - Feature Engineering (változók generálása)  
 FS - Feature Selection (változók kiválasztása)

változókat hagyhatunk ki, ha csak egyszerű heurisztikát alkalmazunk a változók értékelésére és kiválasztására.

A legérdekesebb változó konstruáló módszerek a második stratégiát alkalmazzák, ilyenek például a FICUS algoritmus, ami egyszerű heurisztikán alapul, vagy a FEA-DIS algoritmus és Cognito, amik komplexebbek.

A Python jól ismert **scikit-learn** könyvtára egy olyan függvényt nyújt, amellyel polinomiális változókat lehet generálni (például:  $x^2, x^3$ ), beleértve a közös változókat is (például:  $xy, x^2y^3$ ). Az ilyen polinomiális változók egy részhalmazát képezik, azoknak az új változóknak, amelyeket az **autofeat** generál. Ezek a változók nagyon hasznosak tudnak lenni, gyakran két változó aránya vagy valamelyen kombinációjuk sokkal informatívabb, amelyeket nem tudunk legenerálni a **scikit-learn** módszer segítségével.

A **featuretools** egy népszerű Python könyvtár automatikus változó generálásra, amely egy nagy változóhalmazt generál úgynevezett "mély változó szintézist" alkalmazva. Ez a könyvtár úgy összesíti és traszformálja a változókat, hogy figyelembe veszi a célváltozót. Az **autofeat** is hasonló stratégiát alkalmaz.

Ezeken felül generálhatunk még neplineáris változókat egy alternáló több lépéses folyamat során, amikor a felhasználó választja ki a változótranszformációkat (például:  $\log(x), \sqrt{x}, 1/x, \sin(x), \exp(x)$ ) és kombinálhat változópárokat különböző operátorok segítségével (+, -, ·). Ezek az eredmények exponenciális nagyságrenddel növelik a változók halmazát, azaz például ha kezdetben 3 változónk volt, akkor az első generáló lépés (neplineáris változók traszformálása) után az eredmény körülbelül 20 új változó, a második lépés (változók kombinálása) után 750 és a harmadik lépés (újabb traszformáció) után több, mint 4000 új változó keletkezett. Mivel ez nagymennyiségű RAM-ot igényel a bemeneti változók számától függően, az adatpontok létrehoznak egy almintát az új változók generálása előtt. Gyakorlatban 2-3

### 3.2 Változók kiválasztása

---

ilyen generáló lépés végrehajtása elegendő új, releváns változók eléréséhez.

Az `autofeat` az új változók generálásához a SymPy Python könyvtárat használja, amely automatikusan leegyszerűsíti a generált matematikai kifejezést és ezáltal csökkenti az elvégzendő matematikai műveleteket. Ha az eredeti változók fizikai egységekkel rendelkeznek, akkor csak azok az új változók lesznek megtartva, amelyek megtartják ezeket a fizikai egységeket, tehát például egy hőmérsékletet jellemző változóból nem lesz kivonva egy térfogatot jellemző változó. Ezt a Pint Python könyvtár valósítja meg, amely továbbá számos dimenzió nélküli mennyiséget hoz létre az eredeti változókból a Buckingham  $\pi$ -elméletet alkalmazva. Ha kategórikus változók is szerepelnek az eredeti változók között, akkor ezeket először áttranszformálja több bináris változóvá a megfelelő `scikit-learn` modellt alkalmazva.

## 3.2. Változók kiválasztása

Miután akár több ezer új változó lett generálva (gyakran több, mint ahány adatpontunk van), így elengedhetetlen, hogy csak azokat a változókat válasszuk ki, amelyek jelentőségteljes információval rendelkeznek, amikor egy lineáris modell bemeneleként használjuk fel őket.

**Feature selection definíciója** [3]: Az a folyamat, amely során kiválasztjuk a kulcsváltozók részhalmazát, hogy csökkentsük a tanuló probléma dimenzióját.

A `scikit-learn` könyvtár számos lehetőséget nyújt a változók szelektálására is, mint például az egyváltozós változó pontozás, rekurzív változó elimináció és más modell-alapú változó szelekciós megközelítések. Az egyváltozós változó szelektálós módszerek esetében minden változó egyedi, ami sok korrelált változóhoz vezet, mint amilyeneket az `autofeat` generál. A kifinomultabb változó szelektáló technikák egy külső predikciós modellre támaszkodnak, amelyek együtthatói biztosítják a változók fontosságát. Azonban olyan algoritmusok, mint a lineáris regresszió numerikusan instabillá válnak, ha a változók száma meghaladja az adatpontok (rekordok) számát, ami miatt kevésbé praktikusak az ilyen nagyméretű változóhalmazok esetén, amiket az `autofeat` generál.

Először eltávolítjuk az eredetikkel vagy az egyszerűbb változókkal erősen korreláló változókat, majd egy L1-regularizációra támaszkodó lineáris modellt alkalmazunk egy többlépéses változó szelektálásra.

Az egyedi változók halmaza redundáns információkat nyújthatnak vagy önmagukban nem túl informatívak a célváltozóra nézve, azonban más változókkal kombinálva hasznosnak bizonyulhatnak. Továbbá ahelyett, hogy a célváltozóval való összefüggésük alapján rangsorolnánk a változókat valamelyen kritérium szerint, előnyös egy csomagoló (wrapper) módszert alkalmazni, amely egyszerre több változót hatását is figyelembe veszi, hogy egy igéretes részhalmazt szelektáljon. Erre a `scikit-learn` könyvtár által nyújtott Lasso LARS regressziós modellt és egy L1-regularizációs logisztikus regressziós modellt használunk, amely úgynévezett ritka súlyok alapján dönti el, hogy melyik változó legyen kiválasztva. A módszer a változók kiválasztásakor főként egy zaj szűrős módszerre támaszkodik, ahol a modell az

### 3.3 Az autofeat regressziós és osztályozós függvényei

---

eredeti változókon és néhány "zaj" változón (amely véletlen szerűen generált adatokat tartalmaz, vagy a meglévő adatok lettek összekeverve hozzá) tanult, és csak azokat tartotta meg az eredeti változók közül, amelyeknek a modell által generált együtthatója nagyobb, mint a legnagyobb együtthatója bármelyik zaj változónak.

Amikor függetlenek a változók, akkor igen jól működik, ha egy L1-regularizációs modellel választjuk ki a releváns változókat egy nagy változóhalmazból, amely több változót tartalmaz, mint ahány adatpontunk van. Azonban, amikor egy olyan nagyméretű irreleváns változókból is álló halmazon tanul a modell, mint amit az **autofeat** generál, akkor gyakran nem tudja azonosítani az igazán releváns változókat. Ebből kifolyólag először azonosítunk egy ígéretesnek tűnő változókból álló kezdeti halmazt egy az összes változón tanított L1-regularizációs lineáris modellel és kiválasztjuk a legnagyobb abszolút együtthatójúakat. Ezután a többi változót egyenlő méretű halmazokba osztjuk és a kezdeti halmazzal kombináljuk (úgy, hogy minden kis halmaz kevesebb, mint  $n/2$  változóból álljon) és a modell minden részhalmazból további változókat választ ki. A kiválasztott változó részhalmazokat ezután összekombináljuk és egy másik modell tanítására használjuk, amely alapján meghatározzuk a végső változó halmazt. Annak érdekében, hogy még robusztusabb változóhalmazt kapjunk, ezt a szelektálási folyamatot többszörösen megismételhetjük az adat részhalmazain. A magyarázóváltozók szelekciós futtatásainak változórészehalmazait ezután összekombináljuk és az erősen korreláló változókat kiszűrjük (megtartva azokat a változókat, amelyek a legtöbb futás során ki lettek választva). A többi változóval újra tanítjuk a modellt, hogy kiválasszuk a végső változóhalmazt.

### 3.3. Az autofeat regressziós és osztályozós függvényei

Az **autofeat** könyvtár [2] két modellje alkalmas osztályozásra és regresszióra is. Ez egy olyan módszer, amely a meglévő változókhöz hozzáadott, korábban generált és szelektált új változókkal együtt határozza meg a célváltozót. A célja az, hogy a változók halmazának kibővítésével jobb predikciót adjon a célváltozó értékére. Az osztályozás logisztikus regresszióval, míg a regresszió Lasso LARS regresszióval történik [2]. Mindkét esetben keresztsvalidációt (**CrossValidation**) [6] is alkalmaznak a modellek.

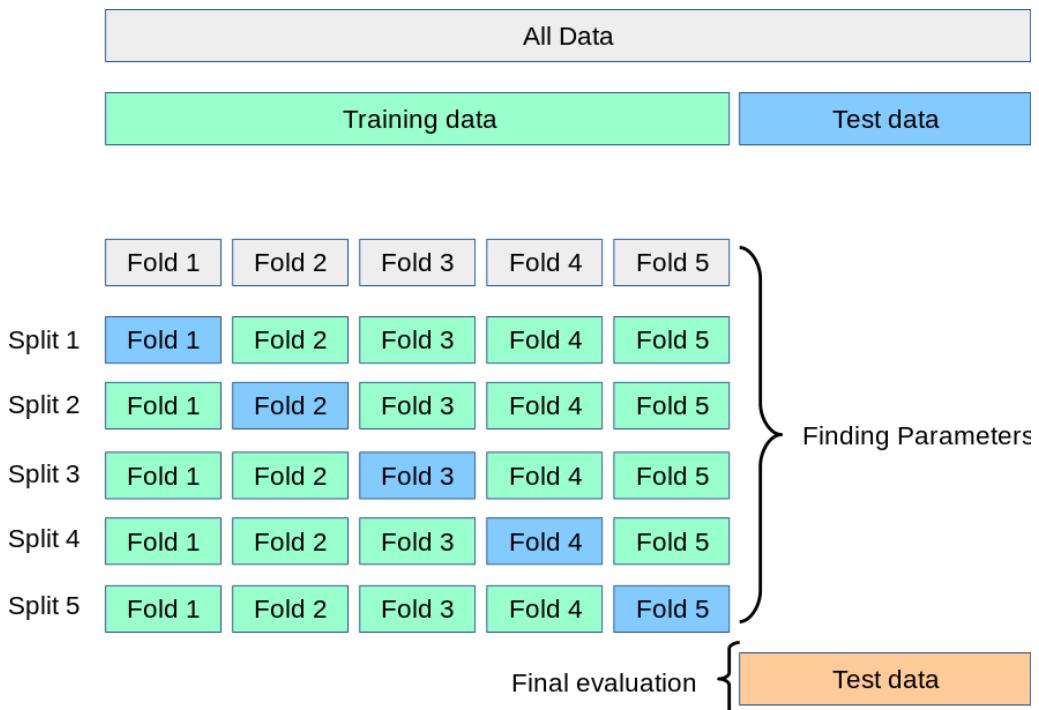
Egy módszertani hiba, ha a predikciós függvény paramétereinek beállítása és tesztelése ugyanazon az adathalmazon történik. Vagyis egy olyan modell, amely csak megismételné a rekordok megtanult címkéit (labels) és tökéletes predikciót adna, azaz nem tudna új adatokon semmilyen érdemleges predikciót végrehajtani, ezt nevezzük túltanulásnak.

Annak érdekében, hogy mindezt elkerüljük, gyakran szétosztjuk az adathalmazunkat tanító- és tesztadatokra, ahol a tanítóadatokon betanítjuk a modelltünket és a tesztadatunkon ellenőrizzük, hogy mennyire teljesít jól (kiválasztjuk a legjobban teljesítőt). Egy másik lehetőség, ha nem kettő, hanem három részre osztjuk az adathalmazunkat, tanító-, validációs- és tesztadatokra, ahol a validációs halmaznak az a feladata, hogy segítségével a modelljeink közül kiválasszuk a legjobban teljesítőt és

### 3.3 Az autofeat regressziós és osztályozós függvényei

csak azt értékeljük ki a tesztadatokon. Ám ebben az esetben elpazaroljuk az akár tesztelésre is használható adatainkat a validáció során.

Erre a problémára a  $k$ -szoros kereszтvalidáció a megoldás. Felosztjuk az adathalmazunkat tanító- és tesztadatokra, majd a tanítóadatokat  $k$  egyenlő részre. A modell betanítására  $k - 1$  darab részhalmazát használjuk a tanítóadatoknak, majd a betanított modellt a maradék 1 részhalmazon validáljuk. Mindezt  $k$ -szor megismerjük, minden másik részhalmazt alkalmazva a validálás során, hogy megtaláljuk a modell megfelelő paramétereit.



3. ábra. 5-szörös kereszтvalidáció.

#### 3.3.1. Osztályozás

Az autofeat osztályozó (`AutoFeatClassifier`) kereszтvalidációs logisztikus regressziós (`LogisticRegressionCV`) módszert alkalmaz, amelynek a matematikai hátterét az **5.2.2. Logisztikus regresszió** alfejezetben ismertetem.

#### 3.3.2. Regresszió

Az autofeat regresszió (`AutoFeatRegressor`) kereszтvalidációs Lasso LARS módszert (`LassoLarsCV`) alkalmaz, amely a lineáris Lasso [6] modellen alapul. A Lasso a ridge regresszióhoz hasonló lineáris modell, amely szintén az együtthatók nagyságát változtatva ad predikciót.

---

Matematikailag, ez egy regularizációs taggal ellátott lineáris modell. A minimálizálandó célfüggvény a következő:

$$\min_w \frac{1}{2n_{rekord}} \|Xw - y\|_2^2 + \alpha \|w\|_1 \quad (4)$$

A Lasso modell így megoldja a legkisebb négyzetek minimalizálását egy  $\alpha \|w\|_1$  "büntető" tag hozzáadásával, ahol  $\alpha$  egy konstans és  $\|w\|_1$  az együttható vektor  $l_1$ -normája.

A Lasso LARS [6] pedig egy olyan Lasso modell, amely a LARS algoritmust használja, és egy pontos megoldást ad, amely darabonként lineáris az együtthatók normája függvényében.

Magasabb dimenziójú erősen korreláló változókat tartalmazó adathalmazok eseténben, gyakran alkalmazunk keresztvalidációt (`LassoLarsCV`) [6], mert pontosabban meghatározza az  $\alpha$  paraméter értékét, még akkor is, ha kevés a rekordok száma a változókhöz képest.

## 4. Az `autofeat` eljárással kapcsolatos kérdések, feltevések

Ebben a fejezetben megfogalmazok négy problémát, kérdéskört, amelyeket az `autofeat` eljárás kapcsán fogok elemezni a dolgozatomban.

Az első feltevésem az, hogy az `autofeat`, feature expansion eljárás bevetése előtt érdemes dimenziócsökkentést végezni. Ennek az a célja, hogy ha a magyarázóváltozók száma magas, akkor elhagyjuk azokat a változókat, amelyek nincsenek hatással a célváltozóra, ezzel elkerülve azt, hogy amikor a meglévő változók közötti transzformációk bevezetésre kerülnek nagyon magasra ugorjon a magyarázóváltozók száma. Erre a célról a Random Forests `feature_importances_` függvényét használom fel, illetve a Boruta algoritmust, amiket az **5.4. Random Forests Feature importance és Boruta algoritmus** című fejezetben fogok ismertetni.

A második kérdés, amire választ kerestem az, hogy vajon ez az `autofeat` feature expansion eljárás, amely a változók közötti nemlineáris összefüggéseket is beépíti, jobb eredményt hoz-e, mint egy úgy nevezett black box eljárás, amely igyekszik megtanulni a nemlineáris kapcsolatokat is. Összehasonlításokat végeztem erre a célról regressziós és osztályozós feladatok esetében is felhasználva Random Forests-et. Jelen dolgozatomban nem foglalkozom a neurális hálókkal, mivel azok hatékonysága nagyban függ a minta nagyságától és hiperparaméterek optimalizálásától. Mivel az `autofeat` feature expansion eljárás megnöveli a magyarázóváltozók számát, ezért még inkább szükségessé válik a nagy mintanagyság a neurális hálózatok használata esetén. Ezt figyelembe véve jelen munkámban Random Forests-et használtam a nemlineáris összefüggések megtanulására alternatívaként. Az eredményeket a **6.**

---

## Eredmények című fejezetben ismertetem.

A harmadik kérdés, amire választ kerestem az az, hogy a többlépcsős eljárás, amit lehetővé tesz az `autofeat`, minden esetben javít-e az eredményeken.

Negyedik kérdéskörben azt próbáltam körüljárni a tesztadatokon, hogy az új, transzformált változók használatával, mennyire lesznek hatékonyabbak, a lineáris regresszió, illetve a logisztikus regresszión túl, más eljárások, mint például regresszió esetében az SVR, Random Forests, illetve osztályozás esetén az SVC, Random Forests.

Ezen problémák, kérdések körbejárására és megválaszolására dolgoztam ki kísérleteket, eljárásokat, amiket a következő fejezetben részletesen ismertetek.

## 5. Az alkalmazott modellek és dimenziócsökkentő eljárások ismertetése

A GitHub-on található `autofeat` [2] könyvtárnál szerepel egy-egy benchmark kód mind a regresszióhoz, mind az osztályozáshoz. Ezeket a kódokat elemezve és kibővítve végeztem saját futtatásokat és teszteléseket, hogy megtudjam mennyire hasznosak az új generált változók.

Először megismerkedtem az `autofeat`-tel az `autofeat_examples.ipynb` Jupyter notebookon keresztül, milyen Python könyvtárakat használ, hogyan működik és mik a be- és kimenetek. Ezután az alábbi két benchmark notebookot vizsgáltam meg:

- `autofeat_benchmark_classification.ipynb`,
- `autofeat_benchmark_regression.ipynb`,

amelyeket saját gépen is futtattam, valamint módosítottam.

Az első benchmark notebook bemeneti adatai a `scikit-learn` három adathalmaza (`iris`, `wine`, `breast_cancer`), melyeket a későbbi futtatásaim során is használtam. Ez a notebook beolvassa az adathalmazokat, az eredeti változóhalmazzal betanít és letesztel 3 modellt (`LogisticRegression`, `SVC`, `RandomForestClassifier`), ezután alkalmazza az `autofeat`-et és kibővíti a változóhalmazokat, majd az új adathalmazokkal betanít és tesztel az előbbiek közül kettő modellt (`LogisticRegression`, `RandomForestClassifier`). Ezt a notebookot kiegészítettem egy harmadik modellel a második tanítási és tesztelési fázisnál beépítettem egy `SVC`-t is, hogy még több összehasonlítást tudjak majd végezni.

A második benchmark notebook bemeneti adatai a `scikit-learn` két adathalmaza (`boston`, `diabetes`) és a UCI [7] oldaláról betöltött három mintaadathalmaz (`concrete`, `airfoil`, `wine_quality`), melyekből a két `scikit-learn`-öst a későbbi futtatásaim során is használtam. Itt is először beolvassa az adathalmazokat, az eredeti változóhalmazon betanítja és leteszteli a 3 modellt (`RandomForestRegressor`, `Ridge`, `SVR`), majd alkalmazza az `autofeat`-et a változóhalmazok kibővítésére, ezután a kibővített adathalmazokkal kettő modellt (`Ridge`, `RandomForestRegressor`)

---

betanít és tesztel. Ezt is kiegészítettem egy harmadik modellel, beépítettem egy SVR-t, hogy itt is több összehasonlítást tudjak majd végezni.

Ezután felhasználva ezt a két kiegészített benchmarkot az említett adathalmazokra létrehoztam még különböző futási kódokat, melyekkel az előző fejezetben megfogalmazott problémákra, kérdésekre kerestem a választ.

Saját Jupyter notebookjaim futási lépései a következők voltak:

1. Az adatok beolvasása.
2. Dimenziócsökkentés Boruta algoritmus vagy Feature importance eljárásokkal.
3. Az adathalmaz kibővítése új változókkal az `autofeat` segítségével.
4. Dimenziócsökkentés Boruta algoritmus vagy Feature importance eljárásokkal.
5. Az új adathalmazok tesztelése a benchmarkok alapján.

Háromféle kódpermutációt készítettem a fenti lépésekkel mind az öt adathalmazon (`boston`, `diabetes`, `iris`, `wine`, `breat_cancer`), a kódok egyharmadánál kihagytam a 2. és a 3. lépéseket, másik egyharmadánál pedig csak a 2. lépést hagytam ki.

A lépések részletesebb leírása:

1. lépés: Az adatok beolvasása

A beolvasáshoz egy a benchmarkban megírt függvényt használtam, amely a `scikit.learn` beépített `load_` függvényét alkalmazza a mintaadatokra, majd kettő `numpy.array()` tömböt készít belőlük (`features` és `labels`).

2. és 4. lépés: Dimenziócsökkentés Boruta algoritmus vagy Feature importance eljárásokkal:

A dimenzió csökkentésre több módszer is használható, én két módszerrel próbálkoztam. Az egyik a Boruta algoritmus (`BorutaPy()` függvény), amely a véletlen erdőkön alapuló változó szelekciós könyvtára a Pythonnak, ezt az **5.4.2. Boruta algoritmus** alfejezetben részletesen ismertetem. A másik eljárás a Feature importance (`randomforest.feature_importances_` függvény), amely szintén a véletlen erdőkön alapul, ám más módszerrel választja ki a változókat, ezt az **5.4.1. Random Forests Feature importance** alfejezetben részletesen ismertetem.

3. és 5. lépés: Az adathalmaz kibővítése új változókkal az `autofeat` segítségével, valamint az új adathalmazok tesztelése a benchmarkok alapján:

Az előző **4. Feature expansion autofeat segítségével** fejezetben részletesen ismertetett `autofeat` segítségével generáltam az új változókat a 3. lépésben, majd az 5. lépésben a fentebb említett benchmarkokban szereplő modelleket tanítottam és teszteltem az új adathalmazokra az `autofeat` alkalmazása előtt és után is.

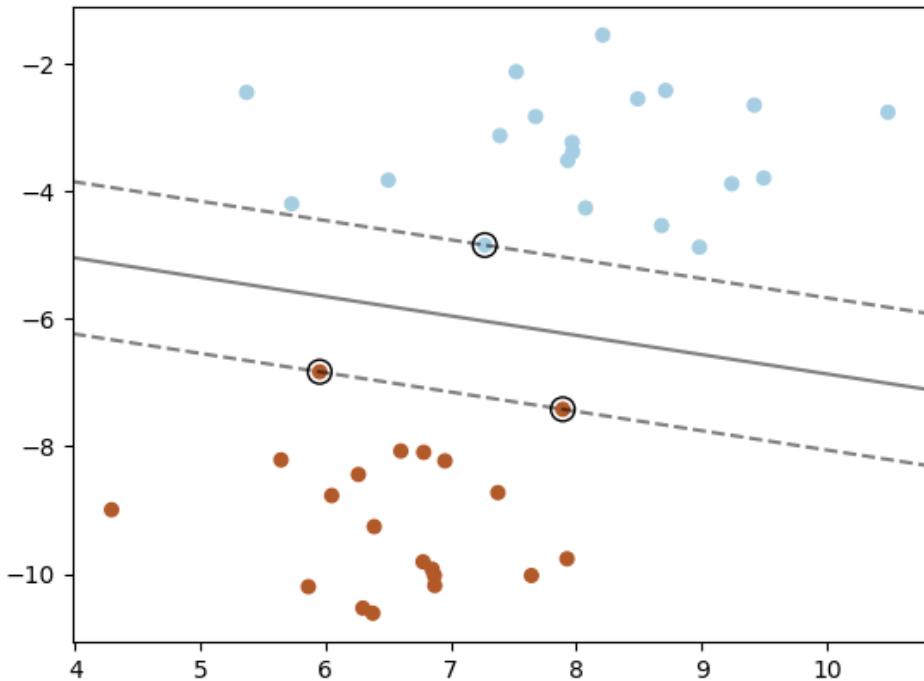
## 5.1 Szupport vektor gépek (SVMs)

---

A következő alfejezetekben szeretném ismertetni a Boruta algoritmus és a Feature importance dimenziócsökkentő eljárásokat, valamint az adatok elemzése során használt regressziós és osztályozási modelleket.

### 5.1. Szupport vektor gépek (SVMs)

A szupport vektor gépek [6] egy csoportja a felügyelt tanulási módszereknek, amelyeket osztályozásra, regresszióra és kilógó értékek megtalálására szoktunk használni. Alapesetben lineáris kétosztályos osztályozásra, szeparálásra képes, de kiterjeszthető többosztályos osztályozásra, nemlineáris szeparálásra, sőt nemlineáris regresszióra is. Az a célja, hogy olyan szeparáló hipersíkot vagy hipersíkokat találjon egy magas vagy végtelen dimenziós térben, amik jól alkalmazhatók osztályozásra, regresszióra vagy más feladatok megoldására. Intuitívan jó szeparáló hipersík az, amelyik a legtávolabb helyezkedik el az elszeparálálandó osztályok legközelebbi elemeihez képest, azaz amelyiknek a legnagyobb az úgynevezett margója (margin), mivel általában a legnagyobb margóval rendelkező hipresíknak a legkisebb az összesített hibája.



4. ábra. Lineáris szeparációs egyenes 3 ponttal, azaz szupport vektorral a margó határán.

## 5.1 Szupport vektor gépek (SVMs)

---

### 5.1.1. Osztályozás (Classification)

A szupport vektor gépek egy része a Support Vector Classification (továbbiakban: SVC) [6] két- vagy többosztályos osztályozásra képes. A döntési függvénye a tanító adathalmaz egy részhalmazától függ, amit szupport vektoroknak nevezünk.

Adottak az  $x_i \in \mathbb{R}^p$ ,  $i = 1, \dots, n$  tanító vektorok, két osztályban és egy  $y_i \in \{1, -1\}^n$  vektor. Az a célunk, hogy megfelelő  $w \in \mathbb{R}^p$  és  $b \in \mathbb{R}$  paramétereket találunk, amelyek egy jó predikciót adnak a  $\text{sign}(w^T \phi(x) + b)$  kifejezéssel a legtöbb ponthoz.

Az SVC a következő primál feladatot oldja meg:

$$\min_{w, b, \zeta} \frac{1}{2} w^T w + C \sum_{i=1}^n \zeta_i, \quad (5)$$

feltéve, hogy:

$$y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \quad (6)$$

$$\zeta_i \geq 0, \quad i = 1, \dots, n. \quad (7)$$

Maximalizálni próbálja a margót (a  $\|w\|^2 = w^T w$  célfüggvény minimalizálásával), miközben bünteti a félreosztályozást vagy ha a pont a margó határán belülre kerül. Ideális esetben az  $y_i(w^T \phi(x_i) + b) \geq 1$  feltétel teljesül minden pontra, ami tökéletes predikciót ad. De nem minden feladat tökéletesen szeparálható egy hipersíkkal, ezért meg kell engedni, hogy néhány pont  $\zeta_i$  költséggel a margó határán belülre essen. Ha a  $C \sum_{i=1}^n \zeta_i$  költségfüggvényben  $C$  nagy, akkor előfordulhat túltanulás.

A primál feladat duálisa a következő:

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha, \quad (8)$$

feltéve, hogy:

$$y^T \alpha = 0, \quad (9)$$

$$0 \leq \alpha_i \leq C, \quad i = 1, \dots, n. \quad (10)$$

ahol  $e$  egy csupa egyesekből álló vektor és  $Q_{ij} \equiv y_i y_j K(x_i, x_j)$  egy  $n \times n$ -es pozitív szemidefinit mátrix, ahol  $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$  a kernel. Az  $\alpha_i$ -ket duális együtthatóknak nevezzük és ezek felülről becsülhetők  $C$ -vel. A duális reprezentáció alapján látható, hogy a tanító vektorok egy magasabb dimenziós térbe történő vetítése nélkül számoljuk ki az egyes vektorok magasabb dimenziós térbeli távolságát a  $\phi$  függvény segítségével, ezt kernel trükknek nevezzük.

Az optimalizációs feladat megoldása után egy adott  $x$  esetén a következőt kapjuk:

$$\sum_{i \in SV} y_i \alpha_i K(x_i, x) + b. \quad (11)$$

## 5.1 Szupport vektor gépek (SVMs)

---

és a megjósolt osztály megfelel a szignum függvény előjelének. Nekünk csak a szupport vektorokat kell összegeznünk, mert minden más vektor  $\alpha_i$  duális együtthatója 0.

### 5.1.2. Regresszió (Regression)

Az SVC módszer kibővíthető regressziós feladatok megoldására is, ezt Support Vector Regression (továbbiakban: SVR) [6] módszernek nevezzük. Hasonlóképpen az SVC módszerhez, az SVR alapján előállított modell is csak a tanító adatok egy részhalmazától függ, mert a költségfüggvény figyelmen kívül hagyja azokat a pontokat, amelyek predikciója közel van a célváltozójukhoz (target).

Adottak az  $x_i \in \mathbb{R}^p$ ,  $i = 1, \dots, n$  tanító vektorok és egy  $y_i \in \mathbb{R}^n$  vektor, az  $\varepsilon$ -SVR a következő primál feladatot oldja meg:

$$\min_{w, b, \zeta, \zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^n (\zeta_i - \zeta_i^*), \quad (12)$$

feltéve, hogy:

$$y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \quad (13)$$

$$w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \quad (14)$$

$$\zeta_i, \zeta_i^* \geq 0, \quad i = 1, \dots, n. \quad (15)$$

Itt büntetjük azokat a pontokat, amelyek predikciója legalább  $\varepsilon$  távolságra vannak a tényleges célváltozójuktól. Attól függően, hogy a pontok predikciója az  $\varepsilon$  sáv fölé vagy alá esik,  $\zeta$  vagy  $\zeta^*$  költséggel szerepelnek a költségfüggvényben.

A duális feladat a következő:

$$\min_{\alpha, \alpha^*} \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \varepsilon e^T (\alpha - \alpha^*) - y^T (\alpha - \alpha^*), \quad (16)$$

feltéve, hogy:

$$e^T (\alpha - \alpha^*) = 0, \quad (17)$$

$$0 \leq \alpha, \alpha^* \leq C, \quad i = 1, \dots, n. \quad (18)$$

ahol  $e$  egy csupa egyesekből álló vektor,  $Q$  pedig egy  $n \times n$ -es pozitív szemidefinit mátrix,  $Q_{i,j} \equiv K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$  a kernel. Mint ahogy az SVC-nél, a tanító vektorok itt is egy magasabb dimenziós térbe történő vetítése nélkül számoljuk ki az egyes vektorok magasabb dimenziós térbeli távolságát a  $\phi$  függvény segítségével (kernel trükk).

## 5.2 Lineáris Modellek

---

A predikció pedig a következő:

$$\sum_{i \in SV} (\alpha - \alpha^*) K(x_i, x_j) + b. \quad (19)$$

### 5.2. Lineáris Modellek

A lineáris modellek [6] a feliúgyelt tanulási módszerek azon csoportja, amelyek regressziót alkalmazva határozzák meg a célváltozót. Az a céljuk, hogy egy folytonos (kvantitatív)  $\hat{y}$  célváltozó értékét meghatározzák a többi változó értékének segítségével:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \dots + w_p x_p. \quad (20)$$

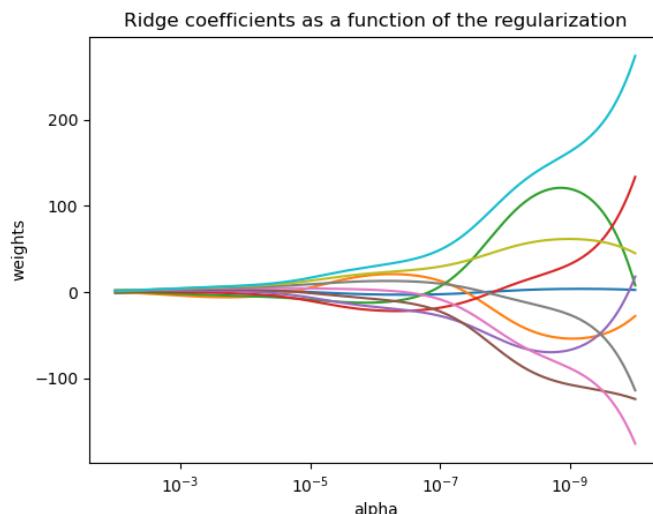
Kihívást jelent a megfelelő  $x_i$  változók kiválasztása és megfelelő modellalak felírása ( $w_i$  együtthatók meghatározása).

#### 5.2.1. Ridge regresszió

A ridge regresszió [6] célja, hogy minimalizálja az átlagos négyzetes hibát az együtthatók nagyságára kiszabott büntetésekkel. A ridge együtthatók minimalizálják a következő büntetett reziduális négyzetösszeget:

$$\min_w \|Xw - y\|_2^2 + \alpha \|w\|_2^2. \quad (21)$$

Az  $\alpha \geq 0$  komplexitási paraméter szabályozza a csökkenés mértékét: ha  $\alpha$  értéke nagy, akkor nagyobb a csökkenés mértéke és az együtthatók közelebb kerülnek egymáshoz és a nullahez.



5. ábra. Ridge együtthatók változása, forrás: [6].

### 5.2.2. Logisztikus regresszió

A logisztikus regressziót (Logistic Regression)[6] a neve ellenére inkább osztályozásra használjuk, mint regresszióra. Olyan néven is ismert, mint például logit regresszió, maximum-entrópiás osztályozás vagy log-lineáris osztályozó. Ebben a lineáris modellben egyetlen próba lehetséges eredményeit jellemző valószínűségeket modellezünk logisztikus függvények segítségével. Bináris vagy többosztályos osztályozás esetén opcionális  $l_1$ ,  $l_2$  vagy Elastic-Net regularizációt alkalmazni.

Mint egy optimalizációs feladat, az  $l_2$  regularizáció paraméterek nullától való eltérésének négyzetével bünteti a paramétereket és minimalizálja a következő költségfüggvényt:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^n \log(\exp(-y_i(X_i^T w + c)) + 1). \quad (22)$$

Hasonlóan az  $l_1$  regularizációs logisztikus regresszió a következő optimalizációs feladatot oldja meg:

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^n \log(\exp(-y_i(X_i^T w + c)) + 1). \quad (23)$$

Az Elastic-Net regularizáció az  $l_1$  és az  $l_2$  kombinációja és a következő költségfüggvényt minimalizálja:

$$\min_{w,c} \frac{1-\rho}{2} w^T w + \rho \|w\|_1 + C \sum_{i=1}^n \log(\exp(-y_i(X_i^T w + c)) + 1), \quad (24)$$

ahol  $\rho$  szabályozza az  $l_1$  és  $l_2$  regularizáció erősségét. Ebben a jelölésben az  $y_i$  célváltozó a  $\{-1, 1\}$  halmazból vehet fel értékeket minden  $i$  esetén. Valamint látható, hogy az Elastic-Net regularizáció ekvivalens az  $l_1$  regularizációval, ha  $\rho = 1$  és ekvivalens az  $l_2$  regularizációval, ha  $\rho = 0$ .

## 5.3. Együttes (ensemble) módszer

Az együttes módszer [6] alapötlete, hogy számos osztályozó modellt összekombinál egy adott tanuló algoritmussal azért, hogy ezáltal javítson osztályozási eljárásban egyetlen osztályozóhoz képest.

Két csoportját szokták megkülönböztetni az együttes módszereknek:

- Átlagolásos módszerek: a fő alapelve az, hogy több független osztályozót építünk és vesszük a predikcióik átlagát. Előnye, hogy a kombinált osztályozó általában jobb, mint bármelyik osztályozó külön-külön, mert a szórása

## 5.3 Együttes (ensemble) módszer

---

így kisebb. Ilyen módszerek például: zsákolásos (bagging) módszerek, véletlen fás erdők, ...

- Ráerősítéses módszerek: ellentétben az előzőekkel, a ráerősítéses módszerek-nél egymás után építjük az osztályozókat és egy megpróbálja csökkenteni a kombinált osztályozó torzítását. A számos gyenge modell kombinálásának motivációja az, hogy előállítsunk egy erőteljes együttes modellt. Ilyen modell például: AdaBoost, Gradient boosting, ...

### 5.3.1. Véletlen erdők

A véletlen erdők (Random Forests) [6] egyike a két döntési fákon (decision trees) alapuló átlagolásos együttes módszereknek. Ez az algoritmus különböző különösen fákra tervezett technikákat perturbál és kombinál össze. Ez azt jelenti, hogy véletlenszerűség bevezetésével jön létre az osztályozók változatos halmaza a konstrukciójuk során. Az egyes osztályozók átlagolt predikciója adja meg az együttes predikciót.

A véletlen erdők esetében minden egyes fához véletlenszerűen kiválasztunk az attribútumok egy részhalmazát, amely jóval kisebb, mint az eredeti tanító attribútum halmaz. Az a célja, hogy csökkentse az osztályozó erdő varianciáját. Az egyes döntési fák általában nagy eltéréseket mutatnak és hajlamosak a túltanulásra. Azzal, hogy a predikciók átlagát vesszük eredményül néhány predikciós hibát megszüntethetünk. A különböző fák kombinációjával csökkenthetjük a varianciát, de néha ez a torzítás növekedésével jár. A jelentős variancia csökkenés pedig jobb modellt eredményez.

A véletlen erdőket használhatjuk osztályozásra (`RandomForestClassifier`) vagy regresszióra (`RandomForestRegressor`). Mindkettő véletlen döntési fákat [6] használ átlagolási módszernek. A döntési fák osztályozásra és regresszióra használt nem paraméterezett felügyelt tanulási módszer. Egy olyan modell készítése a cél, amely az egyszerű döntési szabályok betanulásával meghatározza a célváltozó értékét az attribútumok segítségével.

Adottak az  $x_i \in \mathbb{R}^n$ ,  $i = 1, \dots, l$  tanító vektorok és egy  $y_i \in \mathbb{R}^l$  osztályozó változó, a döntési fa rekurzív módon osztja fel a változók halmazát, hogy az azonos címkével rendelkezők egy csoportba kerüljenek [6].

Legyen az adatunk a döntési fa  $m$  csúcsában  $Q_m$ -el jellemzve,  $N_m$  darab rekorddal. minden egyes lehetséges  $\theta = (j, t_m)$  vágás álljon egy  $j$  változóból és  $t_m$  küszöbértékből, amely az adatot  $Q_m^{bal}(\theta)$  és  $Q_m^{jobb}(\theta)$  részhalmazokba osztja a következők szerint:

$$Q_m^{bal}(\theta) = \{(x, y) | x_j \leq t_m\}, \quad (25)$$

$$Q_m^{jobb}(\theta) = Q_m \setminus Q_m^{bal}(\theta). \quad (26)$$

Egy  $m$  csúcs lehetséges vágásának a "jóságát" egy  $H()$  függvény segítségével (impurity vagy loss function) számítja ki, amelynek meghatározása függ a feladattól

### 5.3 Együttes (ensemble) módszer

---

(osztályozás vagy regresszió):

$$G(Q_m, \theta) = \frac{N_m^{bal}}{N_m} H(Q_m^{bal}(\theta)) + \frac{N_m^{jobb}}{N_m} H(Q_m^{jobb}(\theta)). \quad (27)$$

Olyan paramétereket választunk, amelyek növelik a létrejövő csúcsok homogenitását, azaz növelik a "tisztaságot" (purity):

$$\theta^* = \operatorname{argmin}_{\theta} G(Q_m, \theta). \quad (28)$$

Mindezeket addig ismétli a  $Q_m^{bal}(\theta^*)$  és  $Q_m^{jobb}(\theta^*)$  részhalmazokon, amíg eléri a maximális megengedhető mélységet, azaz  $N_m < \min_{rekord}$  vagy  $N_m = 1$  nem lesz.

#### Osztályozási kritériumok

Ha egy döntési fa  $m$  csúcsában a cél változó a  $0, 1, \dots, K - 1$  értékeket veheti fel, akkor jelölje

$$p_{mk} = \frac{1}{N_m} \sum_{y \in Q_m} I(y = k) \quad (29)$$

az  $m$  csúcsban előforduló  $k$  értékű rekordok relatív gyakoriságát. Egy vágás "jóságát" a következő mérőszámokkal jellemezhetjük:

Gini-index:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \quad (30)$$

Entrópia:

$$H(Q_m) = - \sum_k p_{mk} \log(p_{mk}) \quad (31)$$

Félreosztályozási hiba:

$$H(Q_m) = 1 - \max(p_{mk}) \quad (32)$$

#### Regressziós kritériumok

Ha a cél változó folytonos értékű az  $m$  csúcsban, akkor a következő vágás meghatározásához gyakran minimalizáljuk az átlagos négyzetes hibát (Mean Squared Error - MSE) vagy az átlagos abszolút hibát (Mean Absolut Error - MAE). Az átlagos négyzetes hiba esetében az  $\bar{y}_m$  predikció értéke az átlag, míg az átlagos abszolút hiba esetében  $\operatorname{median}(y)_m$  a medián.

Átlagos négyzetes hiba:

$$\bar{y}_m = \frac{1}{N_m} \sum_{y \in Q_m} y, \quad (33)$$

$$H(Q_m) = \frac{1}{N_m} \sum_{y \in Q_m} (y - \bar{y}_m)^2. \quad (34)$$

Átlagos abszolút hiba:

$$\text{median}(y)_m = \text{median}(y), \quad (35)$$

$$H(Q_m) = \frac{1}{N_m} \sum_{y \in Q_m} |y - \text{median}(y)_m|. \quad (36)$$

## 5.4. Random Forests Feature importance és Boruta algoritmus

A gépi tanulási eljárások esetében gyakori problémát jelent, hogy túl sok változót figyelünk meg, ezek közt gyakran van olyan, amely irreleváns a célváltozóra. Ha túl sok változót használunk egy regressziós vagy osztályozási feladatban, akkor egyszerűen a feldolgozási idő nő, másrészt pedig csökken a pontosság a teszt adaton, vagyis a modell általánosítása romlik [8]. Ezért fontos feladattá vált az adatok előfeldolgozása, amely során arra törekszünk, hogy a lehető legkevesebb magyarázóvaltozót válasszuk ki, amivel elvégezhető a legjobb osztályozás. Ezt a problémát „minimal-optimal” problémának nevezik [9]. Egy másik kapcsolódó probléma a releváns változók felderítése, azoké a változóké, amelyek elősegítik az osztályozást. Ezt "all relevant" problémának nevezik. Ebben az esetben nem a redundancia csökkenése a cél, mint a "minimal-optimal" probléma esetében, hanem a célváltozóra ható változók kiválasztása a feladat.

### 5.4.1. Random Forests Feature importance

A Random Forests beépített `feature_importances_` függvénye, azaz változó fontosság két különböző módszert tartamaz.

Az első eljárás a következő elvre épít. Az egyszerűség kedvéjéről tekintsünk egy döntési fát. Attól függően, hogy a célváltozó egy osztályozó vagy egy folytonos változó, más-más kritériumot használ a döntési cérla. Amikor a célváltozó egy osztályozó, akkor az úgynevezett Gini impurity-t használja, ha pedig a célváltozó folytonos az MSE-t (mean square error) használja. Egy adott változó szerinti döntést követően (két osztályba/halmazba való besorolás) a cél az, hogy két homogén részre bontsuk a döntés alapján a mintaelémeket. Az inhomogenitást diszkrét esetben a Gini impurity segítségével mérjük, folytonos esetben a MSE-vel. A Random Forests az alapján választ döntési változót minden lépésben, hogy mely változó csökkenti a legjobban az inhomogenitást a döntés utáni leszármazott csúcsokban lévő mintaelémek és a szülő csúcsban lévő halmaz között. minden, a döntési fába bekerült változóra ki lehet számítani, hogy átlagosan mennyit csökkentenek az inhomogenitáson. Ha ezt a Random Forests összes fája esetén kiszámoljuk minden egyes változóra, úgy kapjuk meg az egyes változók fontosságát. Ezt az eljárást tartalmazza a `scikit-learn`

## 5.4 Random Forests Feature importance és Boruta algoritmus

---

osztályozós, illetve regressziós feladatra is. A módszer egyik hátránya, hogy preferál olyan diszkrét változók szerint dönteni, amelyek több értéket vesznek fel.

A másik eljárás az out-of-bag mintát használja a fontosság megállapítására. Az eljárás lényege a következő. Több visszatevéses mintát veszünk a fák tanítására. A kiválasztott minta után visszamaradt elemeket nevezzük out-of-bag mintának. Ezeken az elemeken teszteljük majd a megépített erdőt két módon. Az első, hogy átfuttatjuk ezeket a kihagyott mintaelemeket rajta és így megkapjuk a regresszió vagy az osztályozás jóságát, hisz ismerjük ezekre a mintaelemre a célváltozó értékeit. Majd minden egyes változó esetében az out-of-bag tesztelő minta esetén véletlenszerűen permutáljuk az adott oszlop (változó) értékeit. Ezeket a „megzavart” mintaelemeket is áteresztjük a fán. A két eredmény közötti átlagos különbséget fogjuk az adott változó fontosságának nevezni. Ezt a eljárást a `scikit-learn permutation_importance`-nek nevezi. A módszernek gondot okozhatnak ez erősen korrelált változók.

### 5.4.2. Boruta algoritmus

A Boruta algoritmus a Random Forests [10] Feature importance eljárására épít. Az algoritmus alapötlete már Stoppiglia, Dreyfus, Dubois és Oussar 2003-as [11] cikkében jelent meg. A jelenlegi Boruta algoritmust, ami a Random Forests csomagban szerepel Miron B. Kursa és Witold Rudnickinek a 2010-ben megjelent munkájának köszönhető. Az algoritmus neve a szláv mitológiából az erdő öregének, istenének nevéből származik. Feltehetően a Random Forests nevéhez kapcsolódik, hisz a Feature importance score-t használja fel az eljárásban.

Az alapötlet a következő: minden változóhoz hozzárendelnek úgynevezett árnyékváltozókat, amiket az eredeti változókból úgy kapnak meg, hogy véletlenszerűen permutálják az adott változó mintaelemeihez tartozó értékeit. A klasszifikációs eljárásban ezek után, ezeket az új változókat is beleveszi a magyarázóváltozók halmazába. Egy árnyékváltozó fontossága lehet akár pozitív is a véletlennek köszönhetően és ez a fontosság referencia szerepet fog betölteni.

Az algoritmus lépései a következők:

1. Kibővíti a magyarázóváltozók halmazát az árnyékváltozókkal, mindegyik változót legalább 5 másolattal.
2. Véletlenszerűen permutálja az árnyékváltozók értékeit.
3. Feature importance segítségével fontossági Z-score-t rendel az összes változóhoz, az árnyékváltozókat is beleértve.
4. Meghatározza a legmagasabb Z-score-t az összes árnyékváltozó között (MZSA - maximum Z shadow attributes), majd megjelöli azokat a változókat, amelyeknek ennél magasabb a Z-score-juk.

- 
5. Azoknál a változóknál, ahol nehezen dönt, elvégez egy hipotézis vizsgálatot arra nézve, hogy a Z-score szignifikánsan különbözik-e az MZSA-tól.
  6. Azokat a változókat fogja fontosnak tekinteni, amelyeknek a Z-score-ja szignifikánsan nagyobb az MZSA-nál.
  7. Azokat a változókat fogja irrelevánsnak nevezni, amelyek Z-score-ja kisebb lett az MZSA-nál.
  8. Törli az árnyékváltozókat és irreleváns változókat a magyarázóváltozók halma-zából és a előlről kezdi folyamatot.
  9. Addig folytatja ezt az eljárást, amíg nem kell több eredeti változót törölni.
  10. A megmaradt változók lesznek a releváns változók.

Érdemes megjegyezni, hogy a Boruta algoritmus alapvetően egy heurisztika és arra lett kifejlesztve, hogy megtalálja az összes releváns változót, azokat is, amelyek enyhén relevánsak. Nilsson és munkatársai [9] írásában egy változót, akkor neveznek enyhén relevánsnak, ha létezik egy olyan változóhalmaz, amelyben ő nem redundáns változó. Végezetül azt tapasztalták, hogy a futási idő lineárisan nő a magyarázóváltozók számával, illetve akkor sem áll távol a lineáristól, ha a mintaelemek számához képest nézzük.

## 6. Eredmények

A futtatásaim során két regressziós adathalmazon és három osztályozós adathal-mazon hasonlítottam össze az `autofeat` és az említett két dimenziócsökkentő eljárás (Feature importance és Boruta algoritmus) hatékonyságát. Az adathalmazokról összeségében csak annyit szeretnék megemlíteni, hogy nem túl nagy mintaelemszám-mal rendelkeznek (1. táblázat), ezáltal néhol nem túl jó eredményeket produkáltak a modellek, de így legalább a tanításuk és tesztelésük nem igényelt sok időt.

Adathalmaz	Predikció típusa	Sorok	Oszlopok	Predikciós feladat
boston	regresszió	506	13	bostoni lakásárak meghatározása
diabetes	regresszió	442	10	a betegség kiújulásának predikciója
iris	osztályozás	150	4	virágfajták meghatározása
wine	osztályozás	178	13	borok minőségének osztályozása
breast_cancer	osztályozás	569	30	mellrák meghatározása

1. táblázat. Néhány információ a felhasznált adathalmazokról.

Kigyűjtöttem pár generált és leggyakrabban szelektált új változót (2. táblázat) az `autofeat` lépések során (a `feateng_steps` az `autofeat` egy beépített paramétere, amellyel a változók generálásának mélységét lehet beállítani). Megfigyelhető, hogy

---

az első lépés során még egyszerű változókat választ ki, de a későbbi lépésekben már megjelennek a összetett változók is.

	Legyakrabban generált és szelektált új változók
AF1	$1/x$ , $\log(x)$ , $e^x$ , $x^3$ , $x^2$ , $ x $
AF2	$x^3/y$ , $\log(x)/y$ , $x/y$ , $e^x/y$ , $\sqrt{x}/y$ , $x^2/y$ , $x^3y$ , $1/xy$ , $y/\sqrt{x}$ , $e^x \log(y)$ , $x^3 \sqrt{y}$
AF3	$e^x e^y$ , $ x +  y  $ , $ x /x$ , $x/ y $ , $1/(x + y)$

2. táblázat. Leggyakoribb új változók.

AF1 – AutoFeat(feateng\_steps=1)

AF2 – AutoFeat(feateng\_steps=2)

AF3 – AutoFeat(feateng\_steps=3)

A következő öt ábrából az első kettőn (6. és 7. ábra) láthatóak a boston és a diabetes mintaadatokon végzett futtatásaim teszteredményei, ahol az  $R^2$ -et mutatót számoltam ki, majd a másik hármon (8., 9. és 10. ábra) az iris, wine, és breast\_cancer adathalmazokon kiszámolt Accuracy teszteredmények. Az ábrán világoskék színnel jelöltettem az eredeti adatokon végzett tesztelések eredményeit a feltüntetett 3 modellel. Citromsárga színnel jelöltettem, azokat a teszteredményeket, ahol az `autofeat` alkalmazása után, azaz a kibővített változóhalmazokkal a teszteredmények jobbak lettek, mint az eredeti változóhalmazzal végzett teszteket. Zöld színnel jelöltettem azokat a mezőket, amikor az eredeti változóhalmazon még az `autofeat` alkalmazása előtt dimenziócsökkentést végeztem a Boruta algoritmussal és jobb eredményt értem el, mintha nem alkalmaztam volna az algoritmust. Narancssárga színnel pedig azokat a mezőket jelöltettem meg, amikor a még az `autofeat` előtt feature importance segítségével csökkentettem a dimenziót és ezzel jobb eredményt értem el, mintha nem alkalmaztam volna az eredeti változóhalmazon.

A táblázatokban szereplő további jelölések a következők, ahol a data a boston (bos), a diabetes (dia), az iris (iri), a wine (win) és a breast\_cancer (bre) adathalmazokat jelöli, valamint a modell az `AutoFeatRegressor`, `Ridge`, `RandomForestRegressor`, `SVR`, `AutoFeatClassifier`, `LogisticRegression`, `SVC` modellekkel jelöli:

- `benchm` – az eredeti benchmark dokumentumban szereplő teszteredmények
- `data1` – a benchmark saját gépen futtatott eredményei
- `data2` – a tesztelés és az `autofeat` előtt dimenziócsökkentést végeztem
- `data3_af1` – kibővítettem a változóhalmazt `AutoFeat(feateng_steps=1)` alkalmazásával, majd a tesztelés és a további `autofeat` alkalmazása előtt dimenziócsökkentést végeztem
- `data3_af2` – kibővítettem a változóhalmazt `AutoFeat(feateng_steps=2)` alkalmazásával, majd a tesztelés és a további `autofeat` alkalmazása előtt dimenziócsökkentést végeztem

- data4 – elvégeztem egy dimenziócsökkentést, utánna kibővítettem a változóhalmazt `Autofeat(feateng_steps=1)` alkalmazásával, majd a tesztelés és a további `autofeat` alkalmazása előtt dimenziócsökkentést végeztem
- Af1\_modell – a modell tesztelése előtt `AutoFeat(feateng_steps=1)` alkalmazása
- Af2\_modell – a modell tesztelése előtt `AutoFeat(feateng_steps=2)` alkalmazása
- Af3\_modell – a modell tesztelése előtt `AutoFeat(feateng_steps=3)` alkalmazása

input columns	without selection		with boruta selection				with feature_importance selection			
	benchm	bos1	bos2	bos3_af1	bos3_af2	bos4	bos2	bos3_af1	bos3_af2	bos4
	13	13	9	12	19	11	2	4	7	3
Ridge	0,748	0,748	0,743	0,812	0,875	0,813	0,637	0,756	0,817	0,704
RandForRegr	0,871	0,869	0,865	0,862	0,879	0,865	0,800	0,800	0,800	0,800
SVR	0,882	0,882	0,895	0,897	0,827	0,896	0,808	0,802	0,857	0,795
Af1_AutoFeatRegr	0,810	0,816	0,795	0,787	0,876	0,807	0,737	0,786	0,786	0,765
Af1_Ridge	0,802	0,802	0,793	0,808	0,876	0,811	0,756	0,788	0,788	0,784
Af1_RandForRegr	0,862	0,866	0,861	0,860	0,879	0,862	0,800	0,800	0,800	0,800
Af1_SVR	-	0,791	0,905	0,895	0,834	0,891	0,802	0,809	0,809	0,798
Af2_AutoFeatRegr	0,791	0,784	0,869	0,853	0,865	0,849	0,750	0,776	0,776	0,792
Af2_Ridge	0,800	0,785	0,875	0,863	0,821	0,870	0,748	0,780	0,780	0,809
Af2_RandForRegr	0,879	0,875	0,879	0,875	0,825	0,860	0,800	0,800	0,800	0,805
Af2_SVR	-	0,795	0,901	0,903	0,821	0,895	0,810	0,801	0,801	0,792
Af3_AutoFeatRegr	0,048	-0,184	-	-	-	-	0,784	0,046	0,046	0,172
Af3_Ridge	-0,295	-1,098	-	-	-	-	0,780	0,118	0,118	0,483
Af3_RandForRegr	0,855	0,871	-	-	-	-	0,791	0,783	0,783	0,804
Af3_SVR	-	0,842	-	-	-	-	0,810	0,768	0,768	0,786

6. ábra. A boston regressziós adathalmaz  $R^2$  teszteredményei.

A 6. ábrán jól látható, hogy sikerült javulásokat elérni és észrevehető, hogy érdekes új változókat generálni, ahogy a sárgával jelölt Ridge és RandomForestRegressor mutatja az `autofeat` után, 0,046-al és 0,004-el nőtt az  $R^2$ . Még nagyobb növekedést értem el, amikor csökkentettem a dimenziót is, a Boruta algoritmus alkalmazása esetén a legnagyobb növekedés 0,127 volt, míg a Feature importance alkalmazása során 0,069, mindenkor a Ridge regressziós modell esetben. A Boruta algoritmus esetében szinte az összes modell esetében történt javulás, mind a Boruta szelekció nélküli állapothoz képest, mind magán a Borután belül, amikor alkalmaztam az `autofeat`-et. Az is megfigyelhető, mivel kis mintaelemszámú adathalmazról van szó az eredményeket gyakran eltorzíthatják a kilógó értékek, ez történettel a táblázat alsó egynegyedében szereplő negatív adatok esetében is.

A 7. ábrán látható, hogy az  $R^2$ -ek értékei igen alacsonyak, amely az adat boonyolultságára utalhat, lineáris regresszió esetén, nem kapunk jó illeszkedést, vagyis a változók között nemlineáris kapcsolat áll fenn. A táblázat alsó egynegyedében megjelennek negatív értékek is. Ez furcsán hathat, mivel  $R^2$ -ről van szó, viszont az

abból adódik, hogy a determinációs együttható csak lineáris regresszió esetén van 0 és 1 között. Mivel az  $R^2$ -ek eleve alacsonyak voltak, vagyis a változók között nem erős a lineáris kapcsolat az eredmények alátámasztják, hogy hasznos volt a nemlineáris transzformációkat bevezetni, tehát ez is azt mutatja, hogy érdemes új változókat generálni. A sárgával jelölt részeknél AutoFeat (feateng\_steps=1)-et alkalmazva majdnem mindenhol javulás figyelhető meg, valamint a Ridge regresszió esetében mindenkor autofeat után javulás történt. Amikor csökkentettem a dimenziót a Boruta algoritmus alkalmazásával a legnagyobb növekedés 0,124 volt, míg a Feature importance alkalmazása során 0,071, a Ridge és az SVR regressziós modellek esetében.

input columns	without selection		with boruta selection				with feature_importance selection			
	benchm	dia1	dia2	dia3_af1	dia3_af2	dia4	dia2	dia3_af1	dia3_af2	dia4
	10	10	4	4	4	5	2	2	2	2
Ridge	0,383	0,383	0,507	0,359	0,370	0,351	0,351	0,351	0,354	0,351
RandForRegr	0,354	0,354	0,355	0,343	0,301	0,355	0,343	0,343	0,290	0,343
SVR	0,320	0,320	0,285	0,341	0,304	0,312	0,327	0,327	0,323	0,327
Af1_AutoFeatRegr	0,400	0,389	0,330	0,345	0,369	0,340	0,351	0,353	0,353	0,351
Af1_Ridge	0,393	0,386	0,329	0,354	0,370	0,343	0,350	0,351	0,354	0,350
Af1_RandForRegr	0,356	0,348	0,359	0,344	0,301	0,358	0,341	0,340	0,290	0,341
Af1_SVR	-	0,256	0,266	0,308	0,304	0,268	0,327	0,319	0,323	0,327
Af2_AutoFeatRegr	0,353	0,359	0,337	0,356	0,368	0,349	0,355	0,355	0,364	0,355
Af2_Ridge	0,396	0,395	0,324	0,365	0,370	0,348	0,355	0,355	0,363	0,355
Af2_RandForRegr	0,296	0,306	0,329	0,311	0,301	0,331	0,314	0,314	0,292	0,314
Af2_SVR	-	0,289	0,310	0,328	0,289	0,307	0,291	0,291	0,327	0,291
Af3_AutoFeatRegr	-12,399	0,368	0,343	-29,442	-2,154	-23,473	0,237	0,234	-11,049	0,349
Af3_Ridge	-17,264	0,403	0,344	-28,589	-4,049	-25,501	0,237	0,234	-19,738	0,351
Af3_RandForRegr	0,272	0,305	0,316	0,309	0,294	0,311	0,307	0,310	0,264	0,308
Af3_SVR	-	0,314	0,300	0,253	0,253	0,271	0,254	0,250	0,256	0,326

7. ábra. A diabetes regressziós adathalmaz  $R^2$  teszteredményei.

input columns	without selection		with boruta selection				with feature_importance selection			
	benchm	iri1	iri2	iri3_af1	iri3_af2	iri4	iri2	iri3_af1	iri3_af2	iri4
	4	4	4	6	9	8	2	4	5	4
LogRegr	0,967	0,967	0,967	0,900	0,967	0,967	0,967	0,900	0,967	0,967
RandForClas	0,933	0,933	0,933	0,933	0,967	0,967	0,933	0,967	0,933	0,967
SVC	0,967	0,967	0,967	0,933	0,933	0,933	0,967	0,967	0,967	0,967
Af1_AutoFeatClas	0,933	0,933	0,933	0,967	0,967	0,967	0,967	0,967	0,967	0,967
Af1_LogRegr	0,933	0,933	0,933	0,967	0,967	0,967	0,967	0,967	0,967	0,967
Af1_RandForClas	0,967	0,967	0,967	0,967	0,967	0,967	0,967	0,967	0,933	0,967
Af1_SVC	-	0,967	0,967	0,967	0,933	0,933	0,967	0,967	0,967	0,967
Af2_AutoFeatClas	-	0,967	0,967	0,967	0,967	0,967	0,967	0,967	0,967	0,967
Af2_LogRegr	-	0,967	0,967	0,933	1,000	0,967	0,967	0,933	0,967	0,967
Af2_RandForClas	-	0,967	0,967	0,967	0,967	0,967	0,967	0,967	0,933	0,967
Af2_SVC	-	0,933	0,933	0,967	0,933	0,967	0,967	0,967	0,933	0,967
Af3_AutoFeatClas	-	-	-	-	-	-	1,000	0,967	1,000	1,000
Af3_LogRegr	-	-	-	-	-	-	1,000	1,000	1,000	1,000
Af3_RandForClas	-	-	-	-	-	-	0,967	0,967	0,933	0,967
Af3_SVC	-	-	-	-	-	-	0,967	0,967	0,967	0,967

8. ábra. Az iris osztályozós adathalmaz Accuracy teszteredményei.

	without selection		with boruta selection			with feature_importance selection		
	benchm	win1	win2	win3_af1	win4	win2	win3_af1	win4
			13	16	16	5	8	3
input columns	13	13	13	16	16	5	8	3
LogRegr	0,944	0,944	0,944	0,972	0,972	0,972	1,000	1,000
RandForClas	0,972	0,972	0,972	0,972	0,972	0,972	1,000	1,000
SVC	0,972	0,972	0,972	0,972	0,972	0,972	0,972	1,000
Af1_AutoFeatClas	1,000	1,000	1,000	0,972	0,972	1,000	1,000	0,972
Af1_LogRegr	0,972	0,972	0,972	0,972	0,972	1,000	1,000	1,000
Af1_RandForClas	0,972	0,972	0,972	0,972	0,972	0,972	1,000	1,000
Af1_SVC	-	0,972	0,972	0,972	0,972	1,000	0,972	1,000
Af2_AutoFeatClas	-	1,000	-	-	-	1,000	1,000	1,000
Af2_LogRegr	-	0,972	-	-	-	1,000	1,000	1,000
Af2_RandForClas	-	0,972	-	-	-	1,000	0,972	1,000
Af2_SVC	-	1,000	-	-	-	0,972	1,000	1,000
Af3_AutoFeatClas	-	-	-	-	-	0,889	-	1,000
Af3_LogRegr	-	-	-	-	-	0,889	-	1,000
Af3_RandForClas	-	-	-	-	-	1,000	-	1,000
Af3_SVC	-	-	-	-	-	1,000	-	1,000

9. ábra. A wine osztályozós adathalmaz *Accuracy* teszteredményei.

	without selection		with boruta selection			with feature_importance selection		
	benchm	bre1	bre2	bre3_af1	bre4	bre2	bre3_af1	bre4
			30	23	24	23	7	5
input columns	30	30	23	24	23	7	5	7
LogRegr	0,947	0,947	0,947	0,939	0,956	0,956	0,930	0,939
RandForClas	0,939	0,939	0,930	0,939	0,930	0,912	0,904	0,921
SVC	0,965	0,965	0,965	0,965	0,965	0,947	0,921	0,921
Af1_AutoFeatClas	0,982	0,982	0,982	0,982	0,982	0,921	0,912	0,912
Af1_LogRegr	0,930	0,912	0,947	0,947	0,947	0,947	0,930	0,939
Af1_RandForClas	0,947	0,939	0,939	0,939	0,930	0,921	0,921	0,921
Af1_SVC	-	0,965	0,965	0,965	0,965	0,947	0,921	0,921
Af2_AutoFeatClas	-	0,982	0,982	-	-	0,921	0,912	0,921
Af2_LogRegr	-	0,965	0,965	-	-	0,930	0,930	0,939
Af2_RandForClas	-	0,956	0,947	-	-	0,939	0,904	0,912
Af2_SVC	-	0,974	0,965	-	-	0,947	0,921	0,930
Af3_AutoFeatClas	-	-	-	-	-	0,939	0,912	0,921
Af3_LogRegr	-	-	-	-	-	0,947	0,904	0,912
Af3_RandForClas	-	-	-	-	-	0,930	0,921	0,912
Af3_SVC	-	-	-	-	-	0,947	0,912	0,912

10. ábra. A breast\_cancer osztályozós adathalmaz *Accuracy* teszteredményei.

A 8., a 9. és a 10. ábrákon az osztályozó modellek teszteredményei szerepelnek. Ezeknél az adathalmazoknál és modellekknél jól megfigyelhető, hogy kettőt kivéve, minden eredmény 90%-nál is jobb, bármelyik esetet nézzük. Ezért ezen adatokon nagymértékű javulások nem figyelhetők meg, de azért apróbb növekedések észrevehetők. Sok helyen el sikerült érni az 1 egészes pontosságot, ami nagyon jó, mivel tesztadatokról van szó. Ez mindenmellett, hogy hasznosnak bizonyult a Feature importance alkalmazása és az új változók generálása (mert ezekben az esetekben több helyen is 1 egészes értéket értem el), valószínűleg annak is köszönhető, hogy kis-méretű adathalmazokról van szó és könnyen rá tudnak tanulni a modellek. Ennek ellenére szeretném kiemelni mindenből adatnál a legnagyobb növekedéseket: az iris adathalmaz esetében 0,033-al nőtt a pontosság, a wine adathalmaz esetében

0,056-al és a breast\_cancer adathalmaz esetében 0,018-al nőtt. Külön felhívnam rá a figyelmet, hogy a 9. ábrán a Feature importance alkalmazásával egy-két esetet kivéve, mindenhol 1 egészes eredményt értem el.

A következő ábrákon (11., 12., 13., 14. és 15.) a változók mennyiségeinek alakulása látható a dimenziócsökkentő eljárások során, a táblázatok utolsó soraiban azon változók darabszáma található, amennyit majd a modellek használni fognak. Jól megfigyelhető, hogy egy-két kivétellel sikeresen csökkenteni a magyarázóváltozók dimenzióját és az is látható, hogy a Boruta algoritmus és a Feature importance más-más arányokban csökkenti ezeket. Amíg a Boruta algoritmus csak néhány változót hagy el, úgy a Feature importance a legtöbb esetben szinte a felére csökkenti a magyarázóváltozók darabszámát. Szeretném kiemelni a 13. ábrán végzett dimenziócsökkentési próbálkozást, ahol ugyanis eredetileg is csak 4 magyarázó változónk volt, amelyeken az eljárások nemhogy csökkentettek volna, hanem egyes esetekben (amikor végeztem a tesztelések előtt egy `autofeat`-es változó bővítést) még növelték a számukat.

A táblázatokban szereplő további jelölések a következők:

- SEL – dimenziócsökkentő eljárás
- GEN – `autofeat`-tel új változók generálása
  - before – változók száma a dimenziócsökkentő eljárás előtt
  - after – változók száma a dimenziócsökkentő eljárás után
  - generate – `autofeat`-tel generált összes lehetséges változó száma
  - new – `autofeat`-tel generált új változók száma

		with boruta selection				with feature_importance selection			
		bos2	bos3_af1	bos3_af2	bos4	bos2	bos3_af1	bos3_af2	bos4
input columns		13	13	13	13	13	13	13	13
SEL	before	-	-	-	13	-	-	-	13
	after	-	-	-	9	-	-	-	2
GEN	generate	-	61	2645	47	-	61	2645	11
	new	-	4	19	2	-	4	20	3
SEL	before	13	17	32	11	13	17	33	5
	after	9	12	19	11	2	4	7	3

11. ábra. A boston regressziós adathalmaz magyarázóváltozói darabszámának alakulása a dimenziócsökkentő eljárások során.

		with boruta selection				with feature_importance selection			
		dia2	dia3_af1	dia3_af2	dia4	dia2	dia3_af1	dia3_af2	dia4
input columns		10	10	10	10	10	10	10	10
SEL	before	-	-	-	10	-	-	-	10
	after	-	-	-	4	-	-	-	2
GEN	generate	-	42	1495	20	-	45	1495	10
	new	-	6	7	3	-	6	9	2
SEL	before	10	16	17	7	10	16	19	4
	after	4	4	4	5	2	2	2	2

12. ábra. A diabetes regressziós adathalmaz magyarázóváltozói darabszámának alakulása a dimenziócsökkentő eljárások során.

		with boruta selection				with feature_importance selection			
		iri2	iri3_af1	iri3_af2	iri4	iri2	iri3_af1	iri3_af2	iri4
input columns		4	4	4	4	4	4	4	4
SEL	before	-	-	-	4	-	-	-	4
	after	-	-	-	4	-	-	-	2
GEN	generate	-	24	387	387	-	24	387	96
	new	-	2	5	4	-	2	4	4
SEL	before	4	6	9	8	4	6	8	6
	after	4	6	9	8	2	4	5	4

13. ábra. Az iris osztályozós adathalmaz magyarázóváltozói darabszámának alakulása a dimenziócsökkentő eljárások során.

		with boruta selection			with feature_importance selection		
		win2	win3_af1	win4	win2	win3_af1	win4
input columns		13	13	13	13	13	13
SEL	before	-	-	13	-	-	13
	after	-	-	13	-	-	5
GEN	generate	-	73	73	-	73	504
	new	-	3	3	-	3	7
SEL	before	13	16	16	13	16	12
	after	13	16	16	5	8	3

14. ábra. A wine osztályozós adathalmaz magyarázóváltozói darabszámának alakulása a dimenziócsökkentő eljárások során.

		with boruta selection			with feature_importance selection		
		bre2	bre3_af1	bre4	bre2	bre3_af1	bre4
input columns		30	30	30	30	30	30
SEL	before	-	-	30	-	-	30
	after	-	-	23	-	-	7
GEN	generate	-	164	122	-	164	35
	new	-	2	2	-	3	2
SEL	before	30	32	25	30	33	9
	after	23	24	23	7	5	7

15. ábra. A breast\_cancer osztályozós adathalmaz magyarázóváltozói darabszámának alakulása a dimenziócsökkentő eljárások során.

A következő ábrákon (16., 17., 18., 19. és 20.) az összesen legenerált változók és a kiválasztott új változók mennyisége látható az `autofeat` különböző `feateng_steps` beállításai esetén. A `feateng_steps`-ek növelése esetén megfigyelhető, hogy nagyságrendekkel növekednek a generált változók darabszámai, viszont az ezekből kiválasztott új változóké nem. A futások során azt is megfigyeltem, hogy a `feateng_steps`-ek növelése erősen megnöveli a futási időt is, ugyanis nem mindegy, hogy mennyi változó közül kell kiválasztania a releváns új változókat az `autofeat`-nek.

A táblázatokban szereplő további jelölések a következők:

- AF1 – `AutoFeat(feateng_steps=1)` paraméter beállítással
- AF2 – `AutoFeat(feateng_steps=2)` paraméter beállítással
- AF3 – `AutoFeat(feateng_steps=3)` paraméter beállítással
- generate – `autofeat`-tel generált összes lehetséges változó száma
- new – `autofeat`-tel generált új változók száma

		without selection		with boruta selection				with feature_importance selection			
		benchm	bos1	bos2	bos3_af1	bos3_af2	bos4	bos2	bos3_af1	bos3_af2	bos4
input columns		13	13	9	12	19	11	2	4	7	3
AF1	generate	60	61	47	63	94	58	11	20	20	13
	new	6	6	4	4	2	4	2	2	2	3
AF2	generate	10528	2645	1552	2791	6138	2361	82	286	286	178
	new	15	13	25	13	22	9	3	6	6	5
AF3	generate	54631	54958	-	-	-	-	1743	5961	5961	3690
	new	21	27	-	-	-	-	5	5	5	9

16. ábra. A boston regressziós adathalmazon `autofeat`-tel legenerált összes lehetséges változó és a kiválasztott új változók mennyisége.

		without selection		with boruta selection				with feature_importance selection			
		benchm	dia1	dia2	dia3_af1	dia3_af2	dia4	dia2	dia3_af1	dia3_af2	dia4
		input columns	10	10	4	4	4	5	2	2	2
AF1	generate new	45	45	20	20	22	25	10	10	12	18
		2	3	6	5	0	6	2	2	0	2
AF2	generate new	5950	1495	281	239	285	387	69	69	98	69
		8	10	11	3	1	7	3	3	1	3
AF3	generate new	32161	32458	5973	4700	6229	7828	1442	1442	2254	1442
		16	8	7	8	6	8	5	4	4	3

17. ábra. A diabetes regressziós adathalmazon **autofeat**-tel generált összes lehetséges változó és a kiválasztott új változók mennyisége.

		without selection		with boruta selection				with feature_importance selection			
		benchm	iri1	iri2	iri3_af1	iri3_af2	iri4	iri2	iri3_af1	iri3_af2	iri4
		input columns	4	4	4	4	4	4	4	4	4
AF1	generate new	24	24	24	20	51	46	12	20	28	22
		3	3	3	1	0	0	2	1	0	0
AF2	generate new	-	387	387	286	1730	1396	96	286	508	332
		-	2	2	2	3	3	3	2	3	3
AF3	generate new	-	-	-	-	-	-	2031	5807	10665	6701
		-	-	-	-	-	-	7	5	7	5

18. ábra. Az iris osztályozós adathalmazon **autofeat**-tel generált összes lehetséges változó és a kiválasztott új változók mennyisége.

		without selection		with boruta selection			with feature_importance selection		
		benchm	win1	win2	win3_af1	win4	win2	win3_af1	win4
		input columns	13	13	13	16	16	5	8
AF1	generate new	73	73	73	91	91	27	43	16
		3	3	3	1	0	2	0	1
AF2	generate new	-	3677	-	-	-	504	1295	179
		-	13	-	-	-	9	11	4
AF3	generate new	-	-	-	-	-	10855	-	3242
		-	-	-	-	-	6	-	5

19. ábra. A wine osztályozós adathalmazon **autofeat**-tel generált összes lehetséges változó és a kiválasztott új változók mennyisége.

		without selection		with boruta selection			with feature_importance selection		
		benchm	bre1	bre2	bre3_af1	bre4	bre2	bre3_af1	bre4
input columns		30	30	23	24	23	7	5	7
AF1	generate new	164	164	122	129	123	35	25	34
		3	3	2	1	1	2	1	0
AF2	generate new	-	17085	9502	-	-	758	343	617
		-	12	9	-	-	4	0	2
AF3	generate new	-	-	-	-	-	15478	6945	12990
		-	-	-	-	-	7	3	2

20. ábra. A breast\_cancer osztályozós adathalmazon **autofeat**-tel legenerált összes lehetséges változó és a kiválasztott új változók mennyisége.

#### A 4. fejezetben megfogalmazott kérdések megválaszolása

Az első felvetésem az volt, hogy az **autofeat** feature expansion előtt érdemes-e dimenziócsökkentő eljárásokat alkalmazni. Az eredményeim alapján a legtöbb esetben a dimenziócsökkentő eljárások beszúrása a feature expansion rész elé jelentős javulást mutatott, mind a pontosságra, mind pedig a futási idő csökkenésére nézve.

A második kérdés az volt, hogy az **autofeat** feature expansion eljárás után jobb eredményeket kapok-e, mint előtte. Itt is értem el javulásokat, miután alkalmaztam az **autofeat**-et, több helyen igen szignifikáns volt a változás.

A harmadik kérdés, amire választ kerestem, hogy az **autofeat** többlépcsőssége javít-e az eredményeken. Két esetben figyeltem meg javulást, amikor az **autofeat feateng\_steps** paraméterét 1 és 2 mélységre állítottam, viszont amikor már 3 mélységet alkalmaztam, akkor a nagyon zavaróvá váltak a kilógó értékek, valamint túltanulás történt több modell esetében is. Valamint meg kell említenem, hogy ez a futási időt is a többszörösére növelte.

A negyedik kérdéskör a Support Vektor Gépek és a Random Forests esetét vizsgálja. A Random Forests minden esetben jól teljesített, mind az **autofeat** előtt, mind utána, az SVM esetében már inkább megfigyelhettem javulásokat is az **autofeat** feature expansion eljárás alkalmazása után.

## 7. Összefoglalás

A szakdolgozatom keretén belül megismerkedtem a Python egy számomra újonnan bevezetett könyvtárával [1], **autofeat**-el, amely új változók generálása alkalmas, több regressziós és osztályozós modell matematikai hátterével, valamint a Random Forests Feature importance és Boruta algoritmus dimenziócsökkentő eljárásokkal. A szakdolgozatom készítése közben először megvizsgáltam az **autofeat** könyvtárat, utána jártam, hogy milyen módszert alkalmaz a változók generálására, valamint kiválasztására. Megnéztem milyen elven működik a könyvtár beépített osztályozó és regressziós függvénye. Részletesen tanulmányoztam több regressziós és osztályozó modell matematikai hátterét, működési elvét. Megismerkedtem a Boruta algorit-

mussal és a Feature importance-el, amelyek a változók fontosságának segítségével végeznek dimenziócsökkentést. Mindezek után különböző Jupyter notebookokat írtam, hogy az új könyvtárat és a most mélyebben tanulmányozott modelleket tapasztalati úton is megismerhessem, és különböző következtetéseket, tanulságokat vonhassak le a hasznosságukból. Az `autofeat` eljárással kapcsolatosan az volt az előzetes hipotézisem, hogy érdemes egy dimenzió csökkentési eljárást beiktatni a megkezdése előtt. Ez több szempontból is szerencsésnek bizonyult, egyrészt emiatt a keletkezett változók száma nem nőtt olyan nagyra, ennek köszönhetően a futási időn is csökkentést értem el. A valós példákon való futások legtöbbször alátámasztották, hogy ennek alapján javult a pontosság a teszthalmazon, illetve azáltal, hogy kevesebb magyarázóváltozót használunk, kisebb a rátanulás lehetősége.

## Irodalomjegyzék

- [1] Horn, Franziska and Pack, Robert and Rieger, Michael. Joint European Conference on Machine Learning and Knowledge Discovery in Databases. *The autofeat Python library for automated feature engineering and selection*, Springer: 111-120, 2019.
- [2] <https://github.com/cod3licious/autofeat>
- [3] <https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/create-features>
- [4] Rencberoglu, E. *Fundamental techniques of feature engineering for machine learning*, 2019.
- [5] Press, G. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*, Forbes: 2016
- [6] <https://scikit-learn.org/stable/>
- [7] <https://archive.ics.uci.edu/ml/index.php>
- [8] Kohavi R, John GH. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97: 273–324, 1997.
- [9] Nilsson R, Pe˜na J, Bjørkegren J, Tegnér J. Consistent Feature Selection for Pattern Recognition in Polynomial Time. *The Journal of Machine Learning Research*, 8: 612, 2007.
- [10] Breiman L. Random Forests. *Machine Learning*, 45: 5–32, 2001.
- [11] Stoppiglia H, Dreyfus G, Dubois R, Oussar Y. Ranking a Random Feature for Variable and Feature Selection. *Journal of Machine Learning Research*, 3: 1399–1414, 2003.