Összehasonlító elemzés

Kiss Alex
Miskolczi Miklós

2019. nyár

Tartalomjegyzék

1. fejezet

A feladat

A szakmai gyakorlat második hetét követően a feladatunk az **adatbányászat**tal való megismerkedés, az alapvető algoritmusok működésének megértése volt, valamint ezen módszerek kipróbálása **PAL**[1] és **Python** környezetekben.
A módszerek elsajátítására négy hetünk volt, mely során főleg a hivatalos dokumentációkat olvastuk, mind az **SAP** és mind a szükséges **Python** könyvtárak weboldalán.
Miután a megfelelő *osztályozó*, *klaszterező* és *prediktív* algoritmusokat birtokba vettük, a további feladat egy összehasonlító elemzés elkészítése volt, melyen belül bemutatjuk a két környezet előnyeit, hátrányait néhány algoritmus futtatásával.

[1] Predictive Analysis Library

2. fejezet

Az adathalmaz

A feladatunk megoldásához egy már ”bevált” adathalmazt kellett használnunk, viszont a **PAL** környezet által okozott nehézségek miatt nem volt időnk az adatokat megfelelő alakra hozni, így az Írisz adathalmazra esett a választás. Az Írisz adathalmaz három íriszfajtáról tartalmaz darabonként ötven mintát, melyeknek az alábbi attribútumai ismertek:

- A csészelevelének szélessége centiméterben,
- a csészelevelének hossza centiméterben,
- a sziromlevelének szélessége centiméterben,
- a sziromlevelének hossza centiméterben és
- az adott osztály neve.

Csészelevél hossza (cm)	**Csészelevél szélessége (cm)**	**Sziromlevél hossza (cm)**	**Sziromlevél szélessége (cm)**	**Fajta**
5.1	3.5	1.4	0.2	Iris-setosa
...	...	...	...	...
7.0	3.2	4.7	1.4	Iris-versicolor
...	...	...	...	...
6.3	3.3	6.0	2.5	Iris-virginica

2.1. táblázat. Példa az előfordulásokra

3. fejezet

A környezetek

3.1. PAL

3.1.1. Előkészületek

Ahhoz, hogy egy **PAL** algoritmust végre tudjunk hajtani, rengeteg előfeltétel van. Először is szükségünk van egy **SAP HANA** fiókra, amiért vay fizetünk, vagy *Trial* fiókot hozunk létre, vagy az ingyenes verziót mindenféle megszorítással.
Hogy ha ez megvan, használhatjuk többféleképpen dolgozhatunk:

- Első lehetőségünk az **SAP Web-IDE** használata, aminek sajnos elég sok hátránya van:
  - Elsősorban véges a felhasználható helyünk, ami azt jelenti, hogy nem tudunk sok adatot / táblát egyszerre tárolni, hiszen a kevés hely miatt gyorsan be tud telni.
  - Saját tapasztalatból kiderült, hogy ha egyszerre sokan használják (mint pl egy csapatban), akkor a felhő korlátozza a hozzáféréseket, belassul, így ellehetetleníti a munkát.
  - Végezetül a felső tud instabil lenni; megszakad rengetegszer az adatbázishoz a kapcsolat, ami nélkül szintén nem lehet használni.
- Második lehetőség egy **Express Edition** telepítése saját gépre, ami a fenti negatívumokkal nem rendelkezik, viszont hardver igénye hatalmas egy átlag asztali számítógéphez képest.
- Harmadik megoldás egy szerverhez csatlakozás, amihez már többen is tudnak csatlakozni.

Összefoglalva, már maga az elindulás eléggé bonyolult, viszont ha sikerül egyszer, akkor onnantól kevés probléma van vele (természetesen ha nem a felhőre támaszkodunk).
Természetesen minden környezetet elő kell teremteni, de míg **Python**nál letöltünk n darab csomagot, addig itt jóval több idő előkészíteni a terepet.

3.1.2. Használata adatbányászathoz

3.1.2.1. Sémák létrehozása

Először is létre kell hozni az általunk használt táblák sémáját. Szükségünk van az adatset sémájára, hogy milyen attribútumok lesznek benne. Ezt megadhatjuk manuálisan grafikus tervezővel, illetve **SQL**-scriptekkel.
Míg ez viszonylag kevés attribútummal tökéletesen működik, addig többnél ez rengeteg időt elvesz. Ezt azért említem meg, mert például **Python**nál csak simán betöltjük a táblát és tudjuk használni, nincs feltétlen szükség a sémák megadására.
Miután megvan az általunk megadott séma, utána tölthetjük be az adatokat a táblába. Egy negatívum erről, ami sok bosszúságot okozott nekem hogy sokszor képes felcserélni az oszlopokat. Sémában hiába van megadva a megfelelő sorrend, valamiért a rendes táblában az oszlopok egy másik permutációban szerepelnek. Mivel a hibaüzenetek nem túl beszédesek, ezért sejtelmünk se lehet, hogy ez lenne a baj sok esetben (például algoritmus kiszámolta a helyes outputot, beletöltve a tartalmát a táblába hibát dob és természetesen az algoritmus hibáztatható).
Viszont előny (bár ez minden adatbázis kezelő programra igaz), hogy az általunk létrehozott táblákat (meg egyéb eljárásokat, függvényeket) másik is tudják használni, illetve elérni.

A következő szükséges séma, az a paraméter táblájé. **PAL** algoritmust úgy lehet paraméterezni, hogy van neki egy külön paraméter táblája, amibe be tudjuk szúrni a paraméter nevét és az értékét (ami lehet String, Integer, Double).
Innentől, hogy milyen táblákra van szükségünk az adott algoritmustól függ. Előfordul, hogy további segéd input táblák szükségesek (például **KNN**-nél osztálybeli adatok attribútumjai). Output táblák nem minden esetben szükségesek, például ha csak magát a **PAL** algoritmust **SQL** konzolban hívjuk és nem adjuk meg az outputot, akkor nem kell megadnunk az output tábla sémáját, hanem automatikusan megjeleníti annak eredményét. Természetesen, ha megadjuk akkor tökéletesen elmenti az eredményt.
Bizonyos esetekben hasznos tud ez lenni, például ellenőrzi, hogy jó adatokat ad e az algoritmus, de ha tárolni akarjuk az eredményeket, akkor muszájak leszünk definiálni az output táblákat.
Ahogy sejteni lehet tényleg csak definiálni kell a sémákat, de itt jegyeznénk meg valamit. Ezeket a sémákat kizárólag a dokumentációból lehet megtudni, sok esetben egyértelműen leírja, hogy minek kell lennie. Viszont nem egyszer találkoztam, hogy a leírtak szerint nem működött és nem is sikerült rájönni, hogy

mi a helyes. Illetve néhány algoritmusnál elég bonyolultan adták meg ezeket, emiatt össze lehet keverni nagyon könnyen.

Sajnos elég nagy negatívum, hogy az algoritmusokhoz szükséges sémák eléggé kötöttek:

- Általában első oszlopnak egy id-nek kell lennie, viszont ha nincs, vagy nem az első oszlopban található, akkor nem tudjuk megadni, hogy melyik az.
- Hasonlóképpen adott algoritmusoknál megköveteli hogy az adott oszlop mi legyen, viszont ha nekünk az nem ott van szintén nem tudjuk megadni, hogy melyiket tekintse vannak.
  Természetesen lehet generálni id-t vagy megcserélni az oszlopokat, de elég zavaró tud lenni, hogy nem lehet megadni ezt paraméternek megadni.
- Továbbá csak bizonyos algoritmusoknál lehet attributumokat súlyozni.
- Abban az esetben, ha az adott algoritmus nem rendelkezik ilyen paraméterrel és nem akarjuk, hogy az adott attributummal számoljon, akkor megint hozhatjuk létre az új sémát, tölthetjük fel adatokkal és ez mind időigényes tud lenni, ennél vannak kényelmesebb megoldások máshol.

3.1.2.2. Algoritmusok paraméterezése

Minden algoritmusnak megvannak a saját paraméterei. Ezek befolyásolhatják a működést eléggé, de legtöbbjük csak épphogy.
Van olyan ahol kötelező megadni bizonyos paramétereket (pl DBSCAN-nél automatikusan kiszámolja az epsilon sugarú kört vagy megadjuk mi), de legtöbb esetben nem kell, ilyenkor default értékkel fut le.
Mielőtt meghívnánk a **PAL** eljárást, ezt a táblát kiürítjük, hogy véletlenül se maradjon előzően használt más algoritmusok paraméterei.
Természetesen lehet mindegyik eljáráshoz külön paraméter táblát használni, de fölöslegesen tárolnánk ezeket.
Ezután beszúrjuk az általunk kívánt paramétert és annak megfelelő értékét a megfelelő típus helyére és fel is van paraméterezve a tábla.
Itt megjegyezném, hogy viszonylag sok paraméter található minden egyes algoritmusnál. Nem mondanám, hogy a legjobban testreszabhatóak az algoritmusok, de így is rengeteg módon befolyásolhatjuk. Ilyenek például, hogy *thread* eloszlás, távolságok számítása, attributumok súlyozása, tehát általában tudunk mindig javítani az eljárásokon, akár pontosság, akár műveletigény szempontjátból.

3.1.2.3. Futtatás

Szintén három féle módszer van az adott algoritmus futtatására:

- **SQL** konzolban futtatjuk a kiválasztott **PAL** eljárást a megfelelő paraméterekkel:
  - Output megadása nélkül megjeleníti a kiszámolt értékeket,
  - Output megadásával a megadott táblákba beszúrja az eredményt. Viszont ennek megvannak az előnyei és hátrányai.
    Előny az, hogy gyorsan meg lehet csinálni, hasznos ellenőrzésekre akár.
    Hátránya, hogy jogosultság hiányában használhatatlan, illetve nem tudjuk tárolni az utasításokat, feltéve, ha előtte van utána még szeretnénk valamit csinálni, nyilván feladat függő.
- Másik lehetőség, hogy külön eljárásban eltároljuk ezeket az utasításokat: Itt már nincs választásunk, muszáj eltárolni az eredményeket.
  szerencsére nem kell mindegyik output táblát eltárolni, általában legegyszerűbb eredményeket szükséges. Tehát, ha nem vagyunk kíváncsiak minden egyes részletre, akkor azokkal nem kell törődni.
  Szükségünk van két eljárásra:
  - Egyik egy **JSON** formátumú eljárás, amiben meg kell adni az általunk kínált algoritmus nevét, illetve a hozzá szükséges táblák típusát és adatirányát.
  - Másik egy rendes **Hana SQL** eljárás, amiben abszolút testreszabhatjuk, feltölthetjük a paraméter táblát, kiválasztjuk melyik inputot használjuk és hogy mit tegyünk az outputtal. Hasonló a **PL/SQL**-hez, így igen nagy szabadság áll elöttünk.

  Ezek után már hívható is lesz a saját eljárásunk, amiben persze használhatjuk a **PAL** algoritmusát.

3.1.2.4. Adatok elemzése

Pozitívum, hogy a legtöbb algoritmus igen bő eredményekkel rendelkezik, rengeteg minden ki tud számolni, ha megadjuk ezeket.
Negatívum, hogy a vizualizációra nem igen sikerült megoldást találni. Feltészem, hogy létezik, de simán megjeleníteni már nem lehet. Ahhoz kell egy kalkulációs nézet, megfelelő *join*-ok, stb, de még így se biztos, hogy meg tudja jeleníteni ezeket.

3.2. Python

A **Python** egy általános célú, nagyon magas szintű programozási nyelv, mely egy rendkívül közkedvelt eszköz adatbányászati célokra, mivel számos könyvtárral rendelkezik, valamint rendkívül egyszerű elsajátítani a nyelv használatát.

3.2.1. Előkészületek

Egy **Python** környezet előkészítéséhez a **PAL** igénybevételével ellentétben nem kell regisztrálnunk, majd csatlakoznunk egy szerverhez, hanem csak le kell töltenünk a számunkra megfelelő verziót,(lehetőleg *3.0* vagy annál frissebb változat) valamint egy editort.
Egy másik lehetőség, melyet a szakmai gyakorlat során használtam, a Jupyter Notebook, ami egy webes interaktív környezet, melyet számos adatbányászattal kapcsolatos fórumon előszeretettel ajánlanak.
A legegyszerűbb megoldás az Anaconda *distribution-t* letölteni, mely tartalmazza a **Python-t**, a **Jupyter Notebook-ot** és számos könyvtárat a matematikai számításokhoz és adatbányászati feladatokhoz.
Bármely lehetőséget is választjuk, hasznos, ha telepítjük a **PIP**-et[1] is.

A nyelv használata rendkívül egyszerű és mivel számos útmutató készült róla, így ezt nem részletezném. A témakörhöz kapcsolódó legfontosabb alapok rövid idő alatt elsajátíthatóak, akár csak a legtöbbet használt könyvtárak szintaktikája is.

Habár a natív nyelvi elemekkel is elvégezhető számos adatbányász feladat, a különböző könyvtárak jelentősen megkönnyítik ezeket. Fellelhető rengeteg algoritmus implementációja, adatszerkezetek, adatvizualizációs eszközök, gépitanulást elősegítő könyvtárak, valamint importálható adathalmazok gyűjteménye is.

A négy hét során a legtöbbet az alábbi modulokat használtam:

- **NumPy**, a többdimenziós tömbök és mátrixok támogatásához, valamint rengeteg matematikai függvény használatához,
- **SciPy**, optimalizációhoz, lineáris algebrához, interpolációhoz valamint egyéb magas szintű matematikai számításhoz,
- **scikit-learn** egy gépi-tanuláshoz használható könyvtár, rengeteg osztályozási, regressziós és klaszterező algoritmussal.
  Mivel a társam is *"beépített"* algoritmusokat használt, így az összehasonlítás során a **SciKit** eljárásait vetettük össze a **PAL** algoritmusaival.
- **pandas**, adat-manipulációhoz és adat-analízishez, továbbá tartalmaz számos adathalmazt is,
- **Matplotlib**, az adatok vizualizációjához.

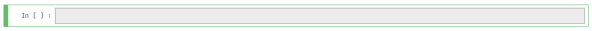
[1] A Python csomag-kezelője.

3.2.2. A Jupyter Notebook használata

Amennyiben megfelelően telepítettük a szükséges programokat, valamint a környezeti változók is megfelelően beállításra kerültek, egy számunkra megfelelő mappából a parancssorból a

```
jupyter notebook
```

paranccsal tudjuk elindítani a webes felületet.
Amennyiben az adathalmazainkat nem könyvtárakból, hanem lokálisan kívánjuk importálni, érdemes ugyanebben a mappában elhelyezni azokat.
Egy új *munkafüzet* létrehozásához kattintsunk a **New**, majd a **Python 3** gombokra. Egy új ablakban megjelenik a létrehozott fájl.



Mint látható, megjelenik egy *"cell"*, avagy egy *"mező"*. A futtatni kívánt parancsainkat ide tudjuk írni. Egy parancs megadása után azt a **Run** gombbal, vagy a **Shift + Enter** billentyűkombinációval tudjuk futtatni.

3.2.2.1. Modulok importálása

A saját modulok importálásához először is létre kell hoznunk azt, majd elmenteni a megfelelő **.py** kiterjesztéssel, és a másik állományban az

```
import <modulnév>
```

paranccsal tudjuk azt elérni. Amennyiben nincs szükségünk az egész modulra, lehetőség van annak csak egy részét, azaz relatívan importálni:

```
from <modulnév> import <rész>
```

A korábban említett modulok használatához először azokat le kell töltenünk, melyhez a már szintén korábban említett **PIP** a legkézenfekvőbb megoldás. A modulokat egy parancssoros felületen az alábbi paranccsal tudjuk letölteni:

```
pip install <modulnév>
```

3.2.2.2. Egy algoritmus futtatása

A **SciKit** modul **K-Közép** algoritmusát például az alábbi módon tudjuk futtatni.

Először importáljuk a szükséges könyvtárakat:

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

Az sklearn/datasets-ből töltsük be az Írisz-adathalmazt, majd a tényleges adatokat egy-egy változóba:

```
iris = datasets.load_iris()
X = iris.data
```

Hozzunk létre egy K-közép objektumot

```
clt = KMeans(n_clusters = 3)
# A klaszterek számát 3-nak választjuk meg.

# Itt tudjuk megadni az algoritmus további paramétereit,
# melyekről a dokumentációban olvashatunk.
```

Ezek után tanítsuk a modellünket:

```
model = clt.fit(X)
```

Az eredményeket az alábbi módon tudjuk megvizsgálni:

```
model.labels_
```

Egy lehetséges eredmény például az alábbi lehet. Az algoritmus hozzárendelte az összes előfordulást egy-egy klaszterhez:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2,
       0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0,
       2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2], dtype=int32)
```

4. fejezet

Az összehasonlított algoritmusok

A szakmai gyakorlat során fellépő hibák miatt habár össze szerettünk volna jóval több algoritmust hasonlítani, a **PAL** esetében csak hármat tudtunk működésre bírni. Ezek a **DBSCAN**, **K-közép módszer** és a **K-legközelebbi szomszédok módszer**. Az algoritmusok összehasonlításához a már korábban említett **Írisz adathalmaz**t használtuk fel. Az algoritmusok futtatása során megfigyeltük a függvények paraméterezhetőségét, a kapott klaszterek elemszámát, valamint a futási időket.

4.1. DBSCAN

A **DBSCAN** implementációk összehasonlítását két kategóriára bontottuk szét az egyik paraméter alapján, miszerint az algoritmus **KD-Tree**-t használva[1], illetve **brute force** módon számítja ki a legközelebbi szomszédot.

4.1.1. Brute force

	PAL	**Python**
Min. samples = 3		
$\epsilon = 1$	50,100	50,100
$\epsilon = 0.75$	50,98 + 2 zaj	49,99 + 2 zaj
$\epsilon = 0.5$	49,84,4 + 13 zaj	49,87 + 14 zaj
$\epsilon = 0.25$	19,13,4,4,5 + 105 zaj	21,35 + 94 zaj
Min. samples = 4		
$\epsilon = 0.5$	24,31,22,41 + 32 zaj	80,50 + 20 zaj
Min. samples = 5		
$\epsilon = 0.5$	49,79 + 22 zaj	45,88 + 17 zaj

4.1. táblázat. A zajok és klaszterek elemszámának változása (Brute force)

4.1.2. KD-Tree

	PAL	**Python**
Min. samples = 3		
epsilon = 0.3	21,6,12,15,8,7 + 81 zaj	23,37 + 90 zaj
epsilon = 0.35	15,7,2,41,12,13,11 + 49 zaj	76,44 + 30 zaj
epsilon = 0.66	42,95 + 3 zaj	48,99 + 3 zaj
Min. samples = 5		
epsilon = 0.5	Nincs észlelhető változás	

4.2. táblázat. A zajok és klaszterek elemszámának változása (KD-Tree)

[1] K-dimenziós fa

4.2. K-közép módszer

A K-közép módszer alkalmazásához is mind a **PAL** és **SciKit** könyvtár számos paramétert kínálnak fel az algoritmus finomhangolásához. A megadható paraméterek többnyire megegyeznek, ám előfordulnak egyes könyvtár-specifikus lehetőségek is, melyet az összehasonlítás végén részleteztünk.

4.2.1. Kezdő klaszterek meghatározása

Az algoritmusok futtatásához **3** klasztert, **100** iterációt, **Euklideszi** távolságot használtunk, az attribútumok súlyozása, valamint normalizálás nélkül.
A **SciKit** algoritmusa két módszert kínál fel, melyek az alábbiak:

- Random
- K-Means++ (gyorsabban konvergál)

A **PAL** pedig az alábbiakat:

- First K observations
- Random with replacement
- Random without replacement
- *Patent of selecting the intial center*

Mind a **SciKit** és **PAL** algoritmusai **168 és 203** ezredmásodpercen belül lefutottak, és a **PAL 50,61 és 39** elemszámú klasztereket állapított meg, kivéve a *random without replacement* opciót, ami **27,26 és 97** elemszámú csoportokat hozott létre. A **SciKit** algoritmusa a **PAL**-hoz hasonlóan **50,61 és 39** elemszámú klasztereket állapított meg. Az iterációk számát **1000**-re növelve a visszatevés nélküli opció is *"beállt"* a többi paraméter esetében kapott értékekre.

4.2.2. Távolságok előszámítása

Csak a **SciKit** algoritmusánál szabályozható, ám ugyanazon paraméterek mellett az eredmények nem változtak.

4.2.3. Használt algoritmus

A **SciKit** algoritmusánál kiválasztható, hogy a *"klasszikus"*, vagy az *"elkan"* variációt használja. Az *"elkan"* lehetőség hatékonyabb, mivel kihasználja a háromszög-egyenlőtlenséget. (Megjegyzendő, hogy ritka mátrixok esetén nem használható, így például *auto* paraméter megadása estén az algoritmus a klasszikus módszert használja.)

4.2.4. Távolságfüggvények

Habár a **Python** környezet és a felhasznált könyvtárak egy sokkal modulárisabb környezetet biztosítanak, így megadhatóak akár a felhasználó által definiált távolságfüggvények is, az összehasonlítás során csak a vizsgált implementációk által biztosított mozgástéren belül maradtunk, így a távolságfüggvényeket kizárólag a **PAL** algoritmusánál tudtuk kiválasztani, melyek az alábbiak, a klaszterek elemszámaival feltüntetve:

- **Csebisev** (50,41,59)
- **Euklideszi** (50,61,39)
- **Manhattan** (50,63,37)

4.3. K-legközelebbi szomszédok módszer

Az algoritmusokat a **KD-Tree** és **Euklideszi-távolság**okkal futtattuk, és a súlyozás szerint osztottuk szét, azon belül az **N-szomszédok** paraméter szerint.

4.3.1. Egyenletes eloszlással

N-neighbors	**PAL**	**Python**
3	53,60,37	53,60,37
5	53,60,37	54,67,29
10	53,60,37	57,63,30

4.3. táblázat. Klaszterek elemszáma különböző N-szomszéd érték mellett

4.3.2. Távolság alapján súlyozva

Amennyiben a távolság szerint súlyozva futtatjuk az algoritmust, a *SciKit* **0.926**-os eredményt ér el, az egyenletes eloszlás **0.830 - 0.853**-as pontjaival ellentétben.
A **PAL** algoritmusa nem mutatott változást a súlyozás hatására, továbbá a **KD-Tree**-t **Brute-force**-ra váltva sem volt tapasztalható számottevő változás egyik környezetben sem.

5. fejezet

Verdikt

Annak ellenére, hogy számos algoritmust a szakmai gyakorlat során fellépő hibák miatt nem volt módunk tesztelni, az adott környezeteknek így is körvonalazódtak az előnyei illetve hátrányai.
Amennyiben az adathalmazainkat egy **SAP HANA** adatbázisban tároljuk, egyértelműen kényelmesebb megoldás a **PAL**-t használni, ám ahhoz hogy a környezetet maximálisan ki tudjuk használni, azt megfelelően kell konfigurálni, ami rendkívül időigényes tud lenni. Továbbá az architektúrából adódik, hogy a számítások rendkívül gyorsak, hiszen az adatainkat memóriában is el tudjuk tárolni, ám hatalmas adathalmazok esetén ez rendkívül költséges lehet.
Ezzel szemben a **Python** környezet konfigurációja sokkal egyszerűbb feladat, valamint a **PAL**-lal ellentétben, a hivatalos dokumentációkon kívül is számos segédanyag fellelhető. További előnye, hogy tudunk a natív nyelvi elemekre illetve az elérhető modulokra is támaszkodni.
Az algoritmusok összehasonlítása során egyértelműen megmutatkozott, hogy a **PAL** egy sokkal robusztusabb környezet, szinte bármilyen eshetőségre fel vannak készítve az algoritmusok, azaz nagyjából bármilyen adathalmaz esetén a megfelelő módon tudtuk a paramétereket finomhangolni. Ezzel ellentétben a **SciKit** implementációi szegényesebb paraméterlistákkal rendelkeztek, például a távolság metrikák megadásánál. Amíg a **PAL** lehetőséget adott a legfontosabb metrikák használatára, addig egyes esetekben a **SciKit** csak az euklideszi távolságot engedélyezte. Más metrikák használatához azokat nekünk kell definiálni, vagy egy már meglévő implementációt tartalmazó modult kell importálni.
Az adatok importálására a HANA környezetben a sémák definiálásánál nem találtunk egyszerűbb opciót, ami nagy adathalmazok esetén rendkívül sok idő. Ezzel ellentétben a Python esetén a **pandas** DataFrame objektumainak használatával elég csak importálni az adatokat például egy .csv állományból.
Az adatok, grafikonok, függvények, dendrogramok illetve egyéb ábrák megjelenítése **Python** környezetben sokkal egyszerűbb, például a **Matplotlib** könyvtár használatával. A **PAL** által kiszámított eredmények, illetve az nyers adatok megjelnítésére Miklós nem talált egyszerű megoldást a **SAP HANA** környezetben.