Szakmai gyakorlat
Hosszú beszámoló

Kiss Alex

2019 nyár

Tartalomjegyzék

1. Bevezető

2. 1-2. hét

2.1. Ismerkedés a környezetekkel

A szakmai gyakorlat első hete főleg megbeszélésekkel telt, melyek során megismerkedtünk a 8 hét alatt teljesítendő feladattal, magával az **SAP HANA** rendszerrel, valamint lehetőségünk volt egy-egy számunkra tetsző modult választani, valamint egy másik modult, arra az esetre, ha az elsődleges modul bármilyen okból kifolyólag el se indulna, illetve a gyakorlat során megszűnne.
Az általam választott modulok az *"adatbányászat **Python** környezetben"* valamint a *"Frond-end fejlesztés **Node.JS / Angular** környezetben"* voltak, így a gyakorlat első két hete során ezekben a témakörökben merültem el.
Már a második nap elején *Béleczki András* egy gyorstalpaló keretében megmutatta hogy hogyan tudunk igénybe venni egy **SAP** fiókot, majd a regisztrációt és bejelentkezést követően bemutatta a **WEB-IDE** használatát.
Ehhez először is a *Szolgáltatások* menüpontban be kellett kapcsolnunk az **SAP Web IDE Full-Stack** opciót, hogy el tudjuk érni a fejlesztői környezetet. Ezt követően a felületet megnyitva, a **File / New / Project from Template** lehetőségre kattintva létrehoztunk egy úgynevezett **SAP HANA Multi-Target-Application**-t, azaz egy **MTA**-t. Ahogyan azt András ajánlotta, érdemes *2.0 SPS 03* vagy annál frissebb verziót használni, így számos lehetséges jövőbeli hibát elkerülve.
A projekt inicializálását követően a felület bal oldalán megjelent a generált mappaszerkezet. Itt a legfontosabb konfigurációs állományokról tartott egy rövid ismertetőt András.
Ezt követően a projektünket kiegészítettük egy adatbázis modullal, melyhez a mappánkra kattintva a **New / Database Module** opciót választottuk, majd a **src** mappában létrehoztunk egy **HDB CDS Artifact** állományt.
Az adatbázis sémáját itt definiáltuk, az **SQL**-ből már megszokott szintaxissal.
A nap hátralévő részében egy hivatalos útmutató segítségével létrehoztam egy **HTML5** modult.
Mivel az elsődlegesen választott modulom az adatbányászat volt **Python** környezetben, és még korábban nem használtam a nyelvet, így nekiláttam az alapokat megtanulni, például a **W3Schools** weboldalán. A nap folyamán sikerült megtanulnom az vezérlési szerkezetek, az alapvető adatszerkezetek, azaz a listák, halmazok és rendezett n-esek használatát, valamint string-műveleteket, mint például a konkatenáció, csere és vágás.

A nap végén a plenáris ülés során a témakör legfontosabb szakkifejezéseit, eszközeit ismertették, így az elkövetkezendő napokban a **Python** és az **SAP HANA** modulok tanulmányozása mellett az alábbi fogalmak / eszközökről olvastam:

- Adattárház és az ehhez kapcsolódó fogalmak:
  - ETL-folyamat;
    - Adattranszformáció
    - Adattisztítás
    - Push / pull adattöltés
  - Adatkocka;
    - Aggregáció
    - Dimenzió,
    - Pivoting,
    - Slicing,
- REST(ful) API-k
  - OData
  - OAuth 1.0, 2.0

Mivel a feladat részét képezte egy front-end modul elkészítése is, valamint mivel a másodlagos modulom a front-end fejlesztés volt, így a hét során a korábban említett témakörök mellett elkezdtem foglalkozni a **JavaScript**-tel, az **Angular**-ral, valamint a **React**-tal is, főleg a hivatalos dokumentációkat, valamint a **Python**-hoz hasonlóan a **W3Schools** weboldalát olvasva.
Habár a pilot projektet követően az elsődleges modulomra fordítottam az időm, a két hét alatt is sikerült megismerkednem a korábban említett technológiák alapjaival, így például egy-egy alkalommal tudtam segíteni az egyik társamnak Angular-ral kapcsolatban.

A pilot projekt előfutáraként megkaptunk egy társaságokról szóló **.csv** állományt, amelyet normalizálnunk kellett, valamint a megfelelő séma létrehozása és finomhangolását követően azt feltölteni az **SAP HANA** környezetünkbe. A hibás előfordulások, a duplikátumok, valamint az érvénytelen attribútumok korrigálását először a webes környezetben szerettem volna végezni, ám mivel ide nem tudtuk feltölteni az állományt, így végül a már korábban megszerzett **Python** tudásommal megpróbáltam megoldani a problémát, egy script segítségével.

2.2. A pilot projekt

A negyedik napon kaptuk meg feladatként, hogy el kell készítenünk a második hét végéig egy pilot projektet, melynek keretében létrehozunk egy, az Országleltár társaságokról szóló weboldalához hasonló oldalt. Ennek kivitelezéséhez létre kellett hoznunk egy back-end és front-end megoldást, a számunkra megfelelő technológiák használatával.

2.3. BingMaps API

A front-end megoldás elkészítéséhez az **Angular**-t ajánlották, így a csapat többi tagjához hasonlóan én is ezt a környezetet választottam. Az Országleltár weboldalán található térképhez hasonlóan nekünk is el kellett készítenünk egy térképet, ami a kiválasztott régiótól, településtől, stb. függően a megfelelő geokódokkal egy **BingMaps** vagy **OpenStreetMaps**-hez hasonló API segítségével megjeleníti a kiválasztott területet.
Ehhez elkezdtem megismerkedni a **BingMaps** API-jával, melynek keretében először is regisztrálnom kellett, hogy kapjak egy kulcsot. Először csak kezdetleges weboldalakat hoztam létre, hogy kipróbáljam az API által kínált lehetőségeket, mint például egy adott terület körvonalazása, vagy egy adott régióra zoomolás, stb.

2.4. OData

Miután a térképes API megfelelőnek bizonyult, elkezdtem utánaolvasni részletesebben a **REST API**-k témakörének, mivel a **SAP HANA** környezetben tárolt adatbázisunkat valamilyen módon össze kellett kötnünk az **Angular** projektünkkel.
Ahhoz hogy a két komponenst összekössük, létrehoztam egy OData szolgáltatást a webes felületen, egy **Node.JS** modulként. Ehhez a webes felületen található varázslót használtam, ami inicializálta a modult. Ezt követően az **xsodata** állományba az alábbi kódrészletet kellett elhelyeznem;

```
service{
"táblanév" as "név";
}
```

ahol a ”táblanév” az eltárolt adatok sémájának a neve, a ”név” pedig az, ahogyan az URL-ben kívánunk hivatkozni rá.
Miután a megfelelő fájl(oka)t újraépítettem, a szolgáltatást futtatni kellett. Az URL-ek megfelelő formázásáról az **OData** hivatalos dokumentációjában olvastam részletesebben.

2.5. Angular projekt

2.5.1. Inicializálás

Mivel a korábbiakban még soha nem dolgoztam **Angular** környezetben, így hasznos volt hogy az előző napok során már elkezdtem foglalkozni vele, így a legalapvetőbb fogalmakkal már tisztában voltam.
A projekt inicializálásához egy megfelelő mappába navigálva egy parancssoros felületen az alábbi parancsokat futtattam:

```
# Egy új projekt inicializálása
ng new <projektnév>

# A projekt elindítása a 4200-as porton
# ng serve --open
```

2.5.2. Az OData beállítása

Mivel eddigre már nem a hivatalos **SAP HANA** szervereket használtuk, hanem egy lokálisat, így módosítani kellett a proxy beállításokat, hogy el tudjuk érni a táblázatainkat. Ehhez az SAP egy hivatalos útmutatóját követtem először, ám a későbbi napok során bejött Vasicsek Gábor és segített megoldani a problémákat, valamint az egyik csapattársammal elkészítettek egy működőképes projektet, ahonnan már mindenki el tudott indulni.

2.6. Egy komponens létrehozása

A weboldalt felépítő komponenseket igyekeztem minél több részre szedni, hogy a későbbiekben egyszerűbb legyen azokat módosítani / karbantartani.
Egy komponens generálásához az alábbi parancsot kell megadnunk:

```
ng generate component <komponensnév>
```

Amennyiben a parancs sikeresen lefut, a mappaszerkezeten belül a **src / app / komponensnév** mappának létre kellett jönnie. Az Országleltár weboldalához hasonlóan, a saját oldalamon elhelyeztem egy táblázatot, amelyet az **OData** szolgáltatáson keresztül az adatbázisban található adatokkal töltöttem fel. Ehhez a HTML állományban az alábbi kódot helyeztem el:

```
<table>
    <tr>
        <th *ngFor="let col of regionColumns">
            {{col}}
        </th>
    </tr>
    <tr *ngFor="let item of regionData">
        <td (click)="selectRegion(item.Regio)">
            {{ item.Regio }}
        </td>
        <td>
            {{ item.Mennyiseg }}
        </td>
        <td>
            {{ item.Ertek}}
        </td>
    </tr>
</table>
```

A **...component.ts** állományba pedig az alábbiakat:

```
// Importok (Itt kell importálnunk az OData komponens is)
@Component({
  selector: 'app-table',
  templateUrl: './table.component.html',
  styleUrls: ['./table.component.scss']
})

export class TableComponent implements OnInit {
    tarsasagok: Observable<any[]>;
    regionColumns: string[];
    regionData: any = [];
    constructor(private atService: TarsasagserviceService,
                private odata: OdataService) {}
```

```
    ngOnInit() {
        this.regionColumns = this.atService.getRegionColumns();
        this.odata.getData(["Regions?$format=json"])
            .subscribe((res: any) => {
            this.regionData = res.d.results;
        });
}
```

Az **OData** komponenst hasonló módon hoztam létre, a

```
ng generate service odata
```

paranccsal. A tartalma az alábbi volt:

```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
@Injectable()
export class OdataService {
    regionData: any = [];
    countyData: any = [];
    settlementData: any = [];
    constructor(private http: HttpClient) { }
    configUrl = "/hana/odata.xsodata/";
    getData(params) {
        return this.http.get(this.configUrl + params);
    }

    ...
}
```

Ezt követően a fő modul **TypeScript** állományát, az **app.module.ts**-t ki kellett egészítenem a megfelelő importokkal. A későbbiekben például a térképhez, illetve egy grafikonhoz a komponenseket hasonló módon hoztam létre.

3. 3-6. hét

A pilot projekt bemutatását követően a szakmai gyakorlat második szakasza a választott modulok témakörének feltárásával telt. Mivel én az adatbányászat témakört választottam, így mielőtt nekiláttam volna a korábbi Python bevezetőkön felbátorodva algoritmusokat implementálni, először utánuk olvastam különböző könyvekből, weboldalakról.
Magáról az adatbányászat témaköréről, valamint az algoritmusokról főként a *Bevezetés az adatbányászatba* (2005, Pang-Ning Tan, Michael Steinbach, Vipin Kumar) című könyvben olvastam, ám az alábbi irodalmak, weboldalak is hasznosnak bizonyultak:

- Tutorialspoint.com
- Guru99.com
- Kaggle.com
- pandas.pydata.org
- Bodon Ferenc, Buza Krisztián - *Adatbányászat*

A harmadik hét során a fentebb említett könyv első nyolc fejezetére fókuszáltam, az adott témaköröknek, algoritmusoknak más forrásokból is utánanéztem. Miután az adatbányászat elméleti hátterével alapszinten megismerkedtem, az algoritmusokat először megpróbáltam implementálni **Python** nyelven. Ehhez először csak egy szöveges editort használtam, viszont miután számos adatbányászattal kapcsolatos fórumon ajánlották a **Jupyter Notebook** környezetet, így telepítettem azt.
A **Jupyter Notebook** egy webes interaktív fejlesztői környezet, ami a forráskódok mellett ábrákat is képes megjeleníteni ugyanazon a felületen, így megkönnyítve az adatok vizualizációját. A telepítéshez az Anaconda Distributiont használtam, így egyszerre hozzájutottam magához a Jupyter Notebook-hoz, valamint számos elengedhetetlen modulhoz, mint például a **NumPy** vagy a **SciKit**. Ezt követően először is megismerkedtem az új környezet használatával a hivatalos dokumentációból, majd a legfontosabb modulok szintaktikájával.

Az adatbányászathoz használt könyvtárak jelentős része a **NumPy** könyvtárra épül, így először ezzel a könyvtárral ismerkedtem meg jobban. A Python által felkínált listákkal ellentétben például a NumPy tömbjei homogének, valamint jóval kevesebb memóriát használnak, továbbá a többdimenziós mátrixok létrehozása is sokkal könnyebb, valamint számos előnnyel rendelkeznek a matematikai számításokhoz.

A **Pandas** (Python Data Analysis Library) az egyik, ha nem a legtöbbet használt eszköz adatanalízishez. Jelentősen megkönnyíti az adatok tárolását és manipulálását, mivel azokat betölthetjük például egy CSV állományból, vagy akár egy SQL adatbázisból és *DataFrame* nevű objektumokban tárolhatjuk őket,

mely a **Python** listáinál sokkal alkalmasabbak adatbányászati feladatokra. Az osztályozáshoz, klaszterezéshez, valamint a regressziók számolásához a **SciKit** könyvtárat használtam.

A **PAL** által kínált megoldások többnyire gyorsabbak, valamint több paraméterrel rendelkeznek, ám a **SciKit** algoritmusai összeköthetőek más könyvtárak, illetve a natív nyelvi elemekkel, továbbá a **Jupyter** notebookjában sokkal egyszerűbb az adatvizualizáció.

Az eredmények megjelenítéséhez a **Matplotlib** könyvtárat használtam fel. A **Pyplot** modul segítségével könnyedén létrehozhatóak függvények, hisztogramok, dendrogramok valamint egyéb ábrák.

Az algoritmusok kipróbálásához *Fischer* **írisz adathalmazát** használtam fel, mely öt attribútumot tartalmaz:

- a csészelevél hossza centiméterben mérve,
- a csészelevél szélessége centiméterben mérve,
- a sziromlevél hossza centiméterben mérve,
- a sziromlevél szélessége centiméterben mérve,
- az osztály (Setosa, Versicolor, Virginica).

Az adathalmaz 150 előfordulást tartalmaz, minden íriszfajtából ötvenet.

Osztályozási algoritmusokból kipróbáltam egyaránt felügyelt és felügyelet nélküli tanulási eljárásokat is. A felügyelt módszerek közül a **K-legközelebbi szomszéd módszert**, a **logisztikus regressziót**, a **naív-Bayes módszert**, valamint a **Szupport vektor gépeket**, a felügyelet nélküli eljárások közül pedig a **K-közép módszert** és a **DBSCAN**-t.

A különböző algoritmusokról először a *Bevezetés az adatbányászatba* című könyvből tájékozódtam, a megvalósításokról pedig a **Kaggle** fórumain, illetve Robert Layton *Learning Data Mining with Python* című könyvéből.

Az algoritmusok kipróbálása során a **PAL**-t használó csapattársammal folyamatosan megosztottuk a tapasztalatokat és összevetettük az eredményeket.

3.1. Vizsgált algoritmusok

3.1.1. Az adathalmaz vizsgálata

Az olvasott szakirodalom és útmutatók alapján megtanultam, hogy mielőtt nekilátnánk egy adott adathalmazból új adatokat kinyerni, mindig érdemes először a kiindulási adatokat megvizsgálni. Az importált adathalmazról a **pandas** modul segítségével lekérdezhető számos tulajdonság.
Az írisz-adathalmazt egy **.csv** állományból importáltam, a

```
dataset = pandas.read_csv('iris.csv')
```

eljárás segítségével.
Ezt követően a *datasets* változóban tárolt adathalmazról lekérdezhetőek annak tulajdonságai:
A `dataset.head()` eljárással lekérdezhetjük az adathalmaz első öt elemét.
A `dataset.shape()` az adathalmaz dimenzióinak méretét adja meg, a `dataset.info()` megadja az attribútumok típusát és megszorításait, a `datasets.describe()` pedig ad egy rövid összefoglalót az adatokról, mint például azoknak a száma, közepe, minimuma, maximuma.

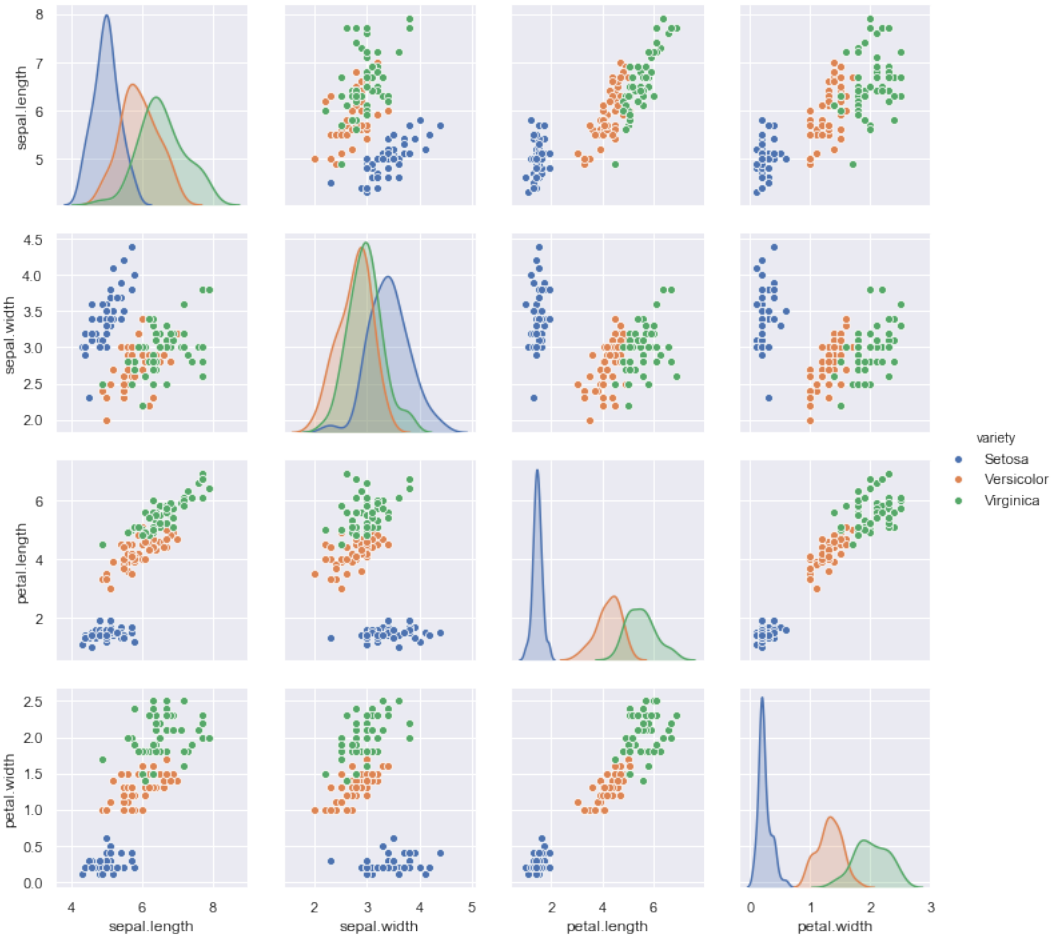
```
In [7]: iris.describe()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

1. ábra. A **describe** által megjelenített információk

A **seaborn** modul használatával könnyedén tudjuk ábrázolni az attribútumok közötti páronkénti kapcsolatokat, például a `seaborn.pairplot(dataset, hue="variety")` paranccsal, ahol a *"variety"* a három osztály szerint színezi az ábrát.

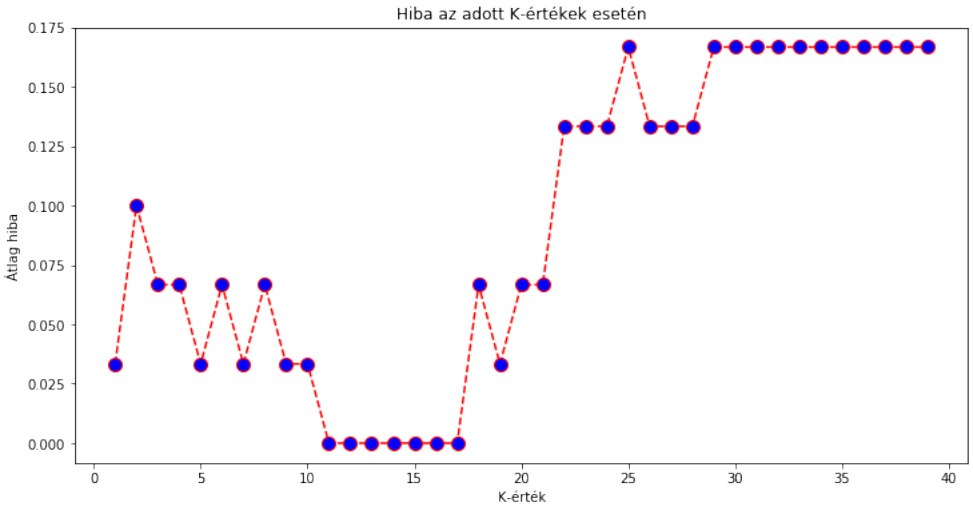


2. ábra. Az adathalmazhoz tartozó *"pairplot"*

Például már most megfigyelhető, hogy a **Setosa** csészelevelének a hossza és a sziromlevelének a szélessége sokkal kisebb mint a másik két osztály példányainak. Ez később felhasználható ha binárisan szeretnénk osztályozni, például logisztikus regresszióval, tehát szét tudjuk bontani a két osztályt Setosa, illetve nem Setosa-ra.

3.1.2. K-legközelebbi szomszédok módszer

A K-legközelebbi szomszéd módszer egy lusta algoritmus, mivel nincs specializált tanítási fázisa, mivel az összes adatot felhasználja arra. Továbbá a **KNN** nem parametrikus algoritmus, azaz semmit sem feltételez az adatokról.
A **KNN** az egyik legegyszerűbb felügyelt gépi tanulási algoritmus. Csak egyszerűen kiszámítja egy új pont távolságát az összes többi tanuló adatponttól, például az euklideszi vagy Manhattan távolsággal. Ezután kiválasztja a K-legközelebbi pontot, ahol K egy egész szám, végül pedig az új pontot hozzáadja ahhoz az osztályhoz, ahová a K adatpontok többsége tartozik.
Mivel a tanítási fázis során nem ismerjük előre hogy melyik K-érték adja a legjobb eredményeket, így érdemes ezeket az értékeket és a hozzájuk tartozó hibákat ábrázolni, majd az optimális mennyiséget használni.

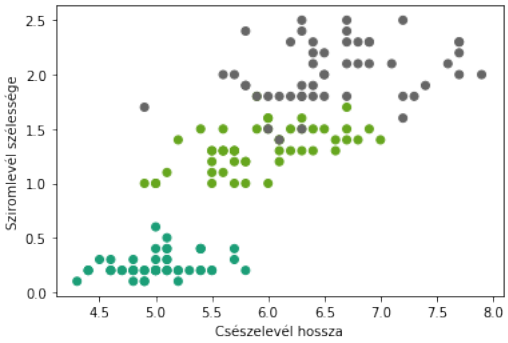


3. ábra. A K-értékek és a hozzájuk tartozó átlag hiba mértéke

Mint az leolvasható, a hiba értéke 11 és 16-os K-érték mellett minimális, így ebben a példában érdemes ebből az intervallumból egy K-értéket választani.

3.1.3. Logisztikus regresszió

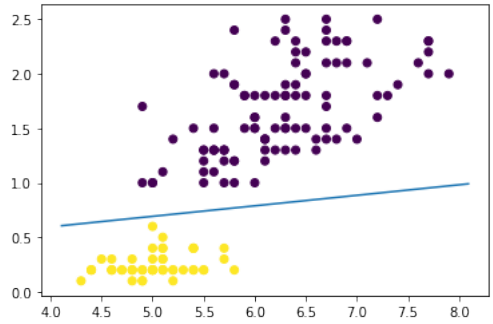
A **logisztikus regresszió** egy statisztikai modell, mely egy adott osztály vagy esemény esélyének meghatározására alkalmas.
A legalapvetőbb formájában egy logisztikus függvényt használ hogy egy binárisan függő változót modellezzen.



4. ábra. A csészelevél szélessége és sziromlevél hossza

Mielőtt elkészítjük a logisztikus regressziós modellt, finomítanunk kell az osztály attribútumon, hogy bináris legyen, mivel a modell bináris módon megállapítja, hogy egy előfordulás a Setosa osztályba tartozik, vagy nem.

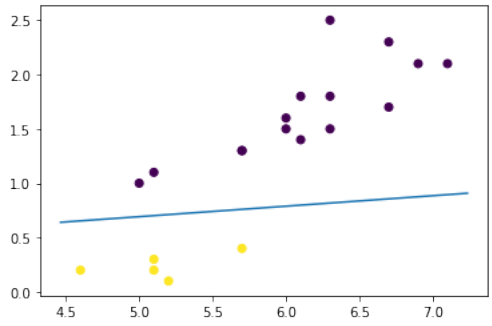
A *train* és *test* halmazokat létrehozva, majd a *train* halmazt a modellnek átadva az alábbi eredményt kaptam:



5. ábra. A két osztály és a döntési határ

Mint látható, a két osztályt könnyedén szét tudtuk választani a döntési határral. A vonal az 50%-os határt jelenti, azaz minél messzebb van egy pont tőle, annál biztosabb hogy az adott osztályba tartozik.

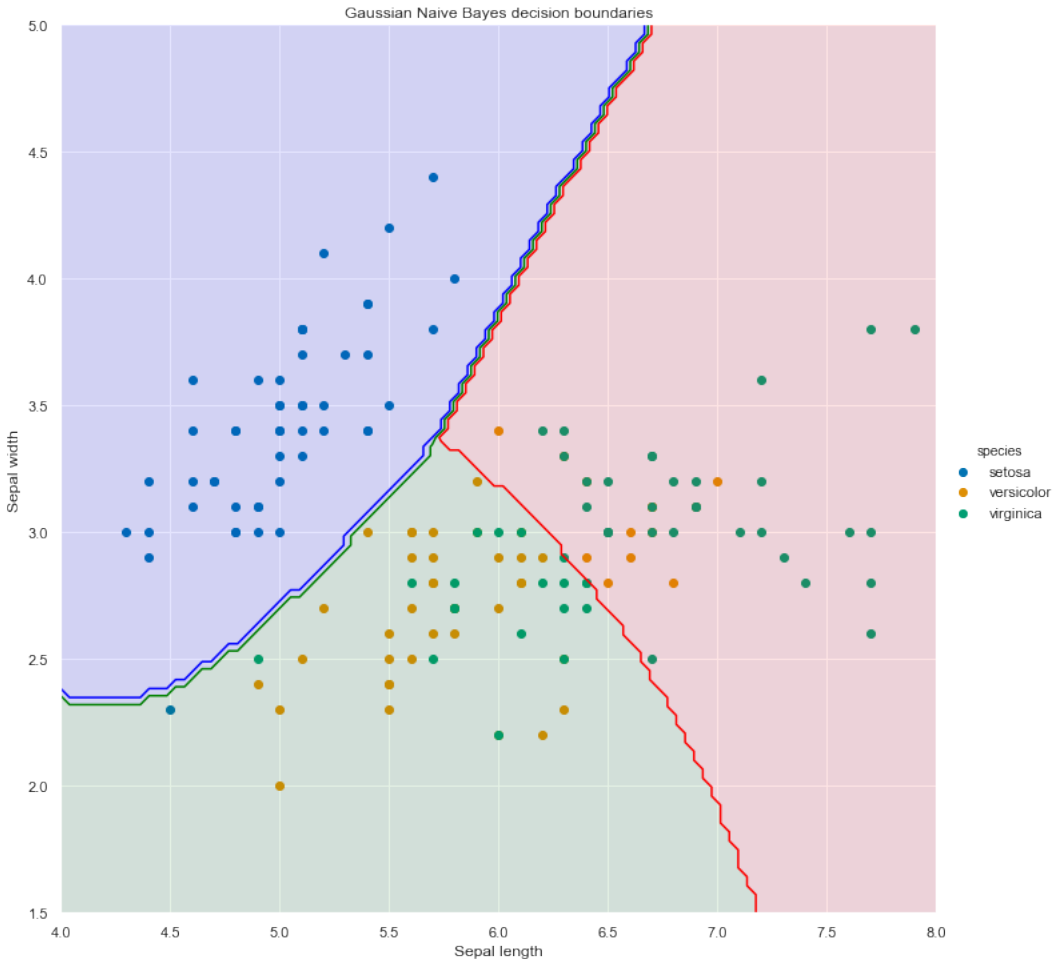
A tanítási fázist követően a maradék példányokra, a *test* halmazra futtatva 100%-os sikerrel el tudta dönteni a modell hogy egy adott példány melyik osztályba tartozik:



6. ábra. 100%-os pontosság

3.1.4. Naív-Bayes módszer

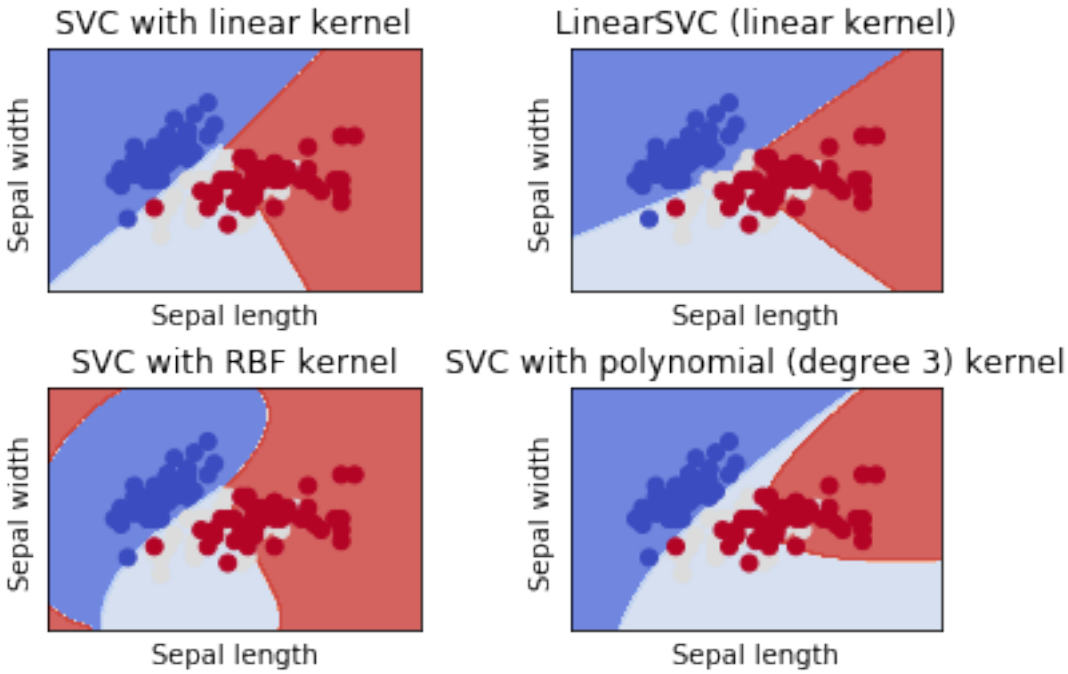
A gépi tanulásban a naív-Bayes osztályozók az úgynevezett egyszerű, "valószínűségi-osztályozók" családját alkotják, a Bayes-tétel alapján, erős (naív) függetlenséggel az attribútumok között.
Az algoritmus a mai napig népszerű szövegosztályozási feladatokhoz, mint például annak eldöntésére, hogy egy levél spam, vagy valódi, a szavak előfordulásának számát, mint attribútumot használva.
A naív-Bayes osztályozók könnyedén skálázhatóak, mivel paraméterei lineárisan nőnek az attribútumok számával együtt.



7. ábra. A naív-Bayes módszer eredménye

3.1.5. Tartóvektor-gépek

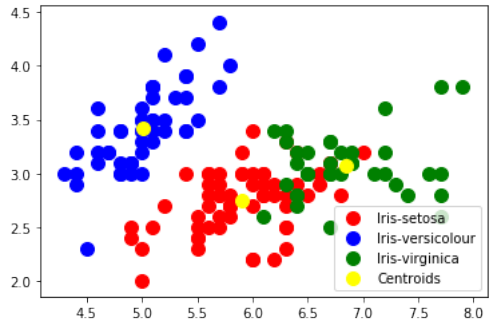
A tartóvektor-gépek a tanulóhalmaz két osztályának valamelyikébe sorolja az új elemeket, így egy *"nem-probabilisztikus"* bináris lineáris osztályozási módszer.
Egy SVM-model a pontok olyan térbeli reprezentációja, ahol az előfordulások egyértelműen elválaszthatóak egy margóval, mely a lehető legszélesebb.
Az újonnan megadott előfordulások a margó adott oldalának megfelelő osztályba fognak tartozni.
A lineáris osztályozáson túl az SVM-ek képesek hatékonyan nem-lineáris osztályozásra is az úgynevezett Kernel-trick segítségével, mely implicit módon hozzárendeli az adatokat egy magasabb-dimenzióbeli térhez.



8. ábra. Az SVM-módszer eredménye különböző paraméterek esetén

3.1.6. K-közép módszer

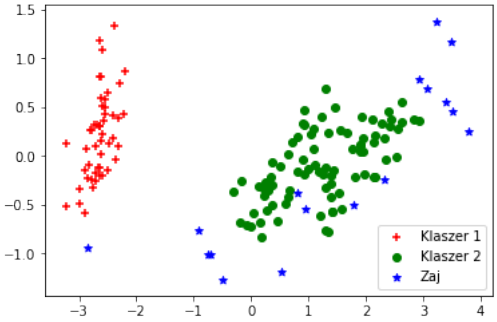
A K-közép az egyik legfontosabb prototípus-alapú klaszterező eljárás. A módszer egy középpontot, vagy úgynevezett *"centroidot"* választ ki prototípusnak, ami általában a pontok egy csoportjának az átlaga.
Első lépésként kiválaszt K darab kezdő középpontot, ahol a K értéke tetszőlegesen megadható, nevezetesen a kívánt klaszterek száma. Ezt követően minden pontot a hozzá legközelebb eső középponthoz rendelünk, így létrehozva a kiinduló klasztereket. Ezután minden klaszter középpontját a hozzárendelt pontok alapján frissítjük, mindaddig amíg egyetlen pont sem frissül, vagy a középpontok ugyanazok nem maradnak.



9. ábra. A K-közép módszer eredménye

3.1.7. DBSCAN

A DBSCAN egy sűrűség-alapú klaszterezési eljárás, ami olyan nagy sűrűségű területeket keres, amelyeket alacsonyabb sűrűségű területek választanak el egymástól. Az algoritmus két bemenő paraméterrel dolgozik, a szomszédsági sugárral és egy sűrűségi korláttal.



10. ábra. A K-közép módszer eredménye

4. 7-8. hét

A szakmai gyakorlat hatodik hete, illetve az utolsó szakasz során az egyik csapattársammal egy összehasonlító elemzést írtunk, melynek keretében összevetettünk egyes adatbányászati algoritmusokat **Python** és **SAP HANA** környezetben. Az algoritmusokat Python környezetben főként a **SciKit** modulból importáltuk, az SAP HANA környezetben pedig az SAP által fejlesztett könyvtárat, a **PAL**-t vettük igénybe.

Az algoritmusok összehasonlítása során figyelembe vettük azoknak az eredményeit, futásidejüket valamint a paraméterezhetőségüket.
Habár a gyakorlat során megvizsgált algoritmusok közül szinte mindegyiket szerettük volna összehasonlítani, a **PAL** használata nehézkesnek bizonyult, így számos algoritmus esetében azokat működésre bírni se tudtuk, különböző problémák miatt.

Az összehasonlításhoz egy már *"bevált"* adathalmazt kellett keresnünk, ám mivel a HANA környezetben minden egyes adathalmazhoz a sémát manuálisan definiálni kell, így a csapattársam úgy döntött, hogy használjunk egy kisebb halmazt, így a már a gyakorlat során is használt írisz-adathalmazra esett a választás.
Az összehasonlítás során megvizsgáltuk a **DBSCAN**-t, a **K-közép módszert**, valamint a **K-legközelebbi szomszédok módszert**, azokat különböző paraméterekkel ellátva.

A DBSCAN-t az algoritmus működése szerint választottuk szét, miszerint egy *KD-Tree*-t használ, vagy *brute-force* módon végzi a számításokat. Paraméterek tekintetében a *min-samples*-t és az *epsilon*-t változtattuk. 3-as min-samples és 1-es epszilon mellett az algoritmusok optimálisan működtek, ám ezeken változtatva a SciKit algoritmusa kisebb epszilon értékekkel a várt 100-50 eloszlásnál rosszabb eredményeket adott vissza. A PAL algoritmusa például 0.25-ös epszilon mellett már 5 klasztert állapított meg, valamint 105 darab zajt.

A K-közép módszer vizsgálata során több paramétert is meg tudtunk vizsgálni, mint például a kezdő klaszterek meghatározásának módját, a távolságok előszámításának engedélyezését, valamint a távolságfüggvények változtatását. Először minden futtatáshoz 100 iterációt, euklideszi-távolságot és 3 klasztert használtunk, majd ezután az iterációk számát növeltük. A PAL esetében 1000-es iterációszám mellett mind a 4 elérhető módszer közel azonos eredményeket adott. A SciKit esetében már kisebb iterációszám mellett is pontosabb eredmények születtek.

A K-legközelebbi szomszédok módszer esetében a különböző N-szomszédok paraméterek esetén hasonlítottuk össze a kapott eredményeket, a DBSCAN-hez hasonlóan szintén KD-Tree és brute-force szerint csoportosítva. Az algoritmusokat eredetileg KD-Tree-re állítva, euklideszi-távolsággal futtattuk, súlyozás sze-

rint szétosztva, az *"n-neighbors"* paraméterrel. A PAL 3,5 és 10-es n-neighbors paraméterrel is ugyanazon eredményeket adta, a SciKit algoritmusa pedig a paraméter értékének növelésével vállt egyre pontosabbá. A KD-Tree-t brute-force-ra cserélve egyik algoritmus sem mutatott számottevő változást.

A szakmai gyakorlat hátralevő ideje alatt az összehasonlítás eredményeit pontosítottuk, valamint elkészítettem az arról szóló dokumentációt, továbbá a gyakorlat során vezetett napló alapján elkészítettem a rövid, illetve a hosszú beszámolót.