



MATEMATIKAI ÉS INFORMATIKAI INTÉZET

Ajánlórendszerek és azok integrálása webalkalmazásokban

Készítette

Besenyei Ferenc

Programtervező Informatikus BSc

Témavezető

Dr. Kovácsnai Gergely

Egyetemi docens

EGER, 2024

Tartalomjegyzék

Bevezetés	5
1. Felhasznált technológiák ismertetése	6
1.1. TypeScript	6
1.2. Statikus kódelemző	7
1.3. Kliens oldali technológiák	7
1.3.1. React	7
1.3.2. Egyéb csomagok, könyvtárak, keretrendszerek	8
1.4. Szerver oldali technológiák	8
1.4.1. Node.js	8
1.4.2. Express	9
1.4.3. Sequelize	9
1.5. Intelligens rendszerben használt technológiák	9
1.5.1. Python	9
1.5.2. Flask	9
1.5.3. Surprise könyvtár	10
1.6. Adatbázis-kezelő rendszer	10
1.6.1. PostgreSQL	11
1.7. Docker	11
1.7.1. Docker Compose	11
2. Gépi Tanulás	12
2.1. Mi az a Gépi Tanulás?	12
2.2. A gépi tanulás lépései	13
2.3. Adat	14
2.3.1. Az adat tisztítása	16
2.3.2. Az adat gyűjtése	17
2.3.3. Az adat készletekre való felosztása	17
2.4. Tanulási típusok	18
2.4.1. Felügyelt tanulás	18
2.4.2. Felügyelet nélküli tanulás	18
2.4.3. Félig felügyelt tanulás	18

2.4.4. Megerősítéses tanulás	19
2.5. Hiperparaméter optimalizálás	19
3. Ajánló rendszerek	21
3.1. Mely problémákra nyújtanak megoldást?	21
3.2. Ajánlási feladat formális leírása	22
3.2.1. Hasznosságfüggvény	22
3.2.2. Tanulási feladat	22
3.2.3. Ajánló rendszer kimenete	22
3.3. Hasonlósági függvények	23
3.3.1. Koszinusz hasonlósági függvény	23
3.4. Különböző megközelítések	24
3.4.1. Kollaboratív szűrés	25
3.4.2. Tartalomalapú módszerek	30
3.4.3. Dimenzió csökkentés alapú ajánlás, mátrix faktorizáció	33
3.5. Az adat előkészítése a modell tanítására és tesztelésre	35
3.5.1. Túllilleszkedés	35
3.5.2. Adatfelosztásos validáció (split-validation)	36
3.5.3. K-szoros keresztszűrések (k-fold cross validation)	36
3.6. Algoritmusok kiértékelése, összehasonlítása és metrikák	36
3.6.1. Pontossági mérőszámok	36
3.6.2. Viselkedési metrikák	39
3.6.3. Online A/B Teszt	40
4. Specifikáció	41
4.1. Feladat specifikáció	41
4.2. Rendszerterv	41
4.2.1. Architektúra	42
4.2.2. Adatbázis terv	43
4.2.3. Film Műfajok	45
4.2.4. Authentikáció	46
5. Fejlesztés bemutatása	48
5.1. Intelligens szolgáltatás Benchmark modulja	48
5.2. Hiperparaméter optimalizálás	50
5.3. Algoritmusok összehasonlítása	52
5.3.1. Újdonság (novelty) értékének kiszámítása	53
5.3.2. Találati arány (hit-rate) kiszámítása	54
5.3.3. A kiértékelés	55
5.4. Adatok tárolása és manipulálása	61

5.4.1. Intelligens szolgáltatás	62
5.4.2. Backend szolgáltatás	64
5.5. Az alkalmazás megtekintése	66
5.5.1. Autentikáció	66
5.5.2. Személyre szabott ajánlások	71
Összegzés	78
Irodalomjegyzék	79

Bevezetés

Az elmúlt időszakban, nem telik el úgy nap, hogy ne jöjjön velünk szembe valamilyen mesterséges intelligenciáról szóló hír, legyen az egy önvezető jármű, nyelvi modell, vagy az első „AI Software Engineer”. Azonban a „mesterséges intelligencia” kifejezés már régóta létezik, kutatják és fejlesztik. Sokkal több felhasználási területe van, mint azt egy átlagember gondolná és sokkal többször is találkozik vele. A technológia fejlődésének hála, ma már „ mindenki ” zsebében ott lapul egy *okostelefon* és elérhető számára az *internet*. Nem is kell az interneten böngészünk, már a számítógép operációs rendszer telepítése alatt, a telepítő engedélyt kér, hogy adatokat gyűjtsön viselkedésünkről és *személyre szabott* hirdetésekkel bombázhasson. Ugyanígy majd minden weboldal azzal kezd, hogy engedélyünket kéri, *cookie*-k formájában adatokat gyűjthessen rólunk hasonló célokra. De ott vannak a különböző szociális- és videómegosztó oldalak *hírfolyamai*, melyeket minden nap látogatunk, és még sorolhatnám.

A szakdolgozatom célja, hogy egy átfogó képet nyújtsak a *gépi tanulásról* és az ehhez kapcsolódó különböző területekről. Ugyanígy, az *ajánlórendserek* témakörét is szeretném bővebben kifejteni, azok különböző megközelítéseit, módszereit, és azt, hogy hogyan tudjuk mindezt megvalósítani és integrálni egy modern webalkalmazásban.

1. fejezet

Felhasznált technológiák ismertetése

Az alkalmazás – ami megtalálható a <https://github.com/fepu08/szakdolgozat> GitHub tárolóban az **mvp** mappában – egy elosztott rendszer, melyet három részre, szolgáltatásra bontok:

- *Frontend szolgáltatás*: A böngészőben látható weblap. A felhasználó számára ez a kezelőfelület.
- *Backend szolgáltatás*: Többek között a frontendet és az intelligens szolgáltatást köti össze.
- *Intelligens szolgáltatás*: Itt készíti az alkalmazás az ajánlásokat.

A rendszerek közötti kommunikációt *HTTP* protokollon keresztül valósítom meg. Bővebben majd a 4.2.1. fejezetben, először hadd mutassam be az alkalmazáshoz felhasznált különböző technológiákat.

1.1. TypeScript

A *TypeScript*¹ egy ingyenes és nyílt forráskódú, a *Microsoft* által fejlesztett magas szintű programozási nyelv, ami statikus típusossággal és opcionális típusannotációkkal egészíti ki a *JavaScript*-et, annak *szupersetje*, tehát minden *JavaScript* program szintaktikailag érvényes *TypeScript* kód. Egyaránt használható kliens- és szerver oldalon is, többek között a *Node.js* használatával.[1] Fordítási időben képes hibát dobni, ezáltal gyorsabb visszacsatolást biztosít a fejlesztők számára. Valamint az 1.2. fejezetben említett *ESlint*-tel együtt használva, a fejlesztés alatt, valós időben láthatjuk a hibát a fejlesztőkörnyezetben. Igazából ez ugyanúgy működik, ahogy *VisualStudio* jelzi a hibát, ha egy *integer* változónak egy *string*et adunk értékül.

¹ TypeScript hivatalos oldala: <https://www.typescriptlang.org/>

1.2. Statikus kódelemző

Statikus kódelemzéshez az *ESLint*²-et használom kliens- és szerver oldalon egyaránt. Jól működik a *TypeScript*-tel és *Prettier*³-rel. Utóbbival együtt használva szép, egységesen formázott kódot kapunk végeredményként.

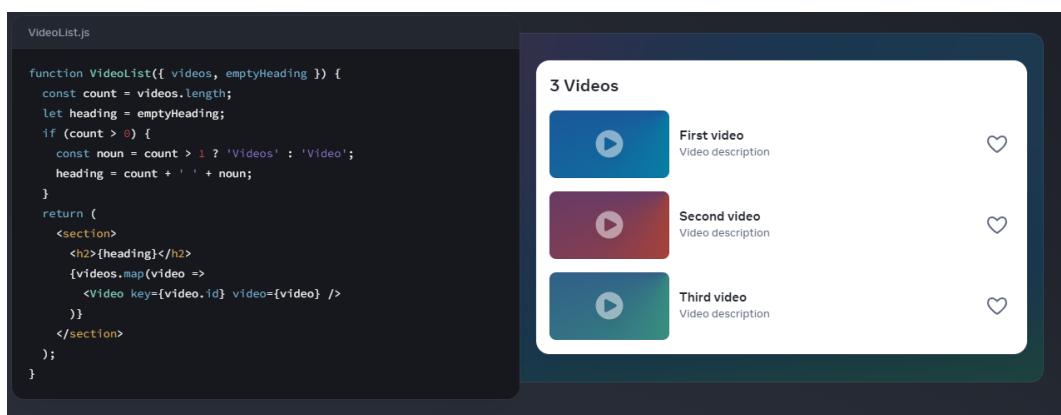
1.3. Kliens oldali technológiák

A frontendünk egy *egylapos webalkalmazás* (single page application vagy SPA) mely megépítéséhez a *React* könyvtárat használtam.

1.3.1. React

A *React*⁴ egy ingyenes és nyílt forráskódú kliensoldali *JavaScript* könyvtár, melyet komponensalapú felhasználói felületek létrehozásához használnak. A *Meta* (korábban *Facebook*), valamint *open-source fejlesztők* által fejlesztett és üzemeltetett.[2]

Lényegében azért választottam, mert népszerű, és lehetővé teszi a komponensalapú fejlesztést, valamint nem utolsó sorban ezzel van tapasztalatom. Az 1.1. ábrán látható, hogy mit is takar a komponensalapú fejlesztés. Az ábra bal felében a *VideoList* nevű komponens programkódja látható, mely böngészőben való megjelenítésé az ábra jobb felén helyezkedik el. Nem mennék bele a részletekbe, viszont számomra nagy előny ebben a koncepcióban, hogy a *React* egységbe zárja a komponenst és az ahhoz tartozó üzleti logikát, emellett ezek a komponensek újrafelhasználhatóak.



1.1. ábra. React - Komponensalapú fejlesztés⁵

² ESLint hivatalos oldala: <https://eslint.org/>

³ Prettier hivatalos oldala: <https://prettier.io/>

⁴ React hivatalos oldala: <https://react.dev/>

⁵ Forrás: <https://react.dev/>

1.3.2. Egyéb csomagok, könyvtárak, keretrendszerek

Mivel nem vagyok túl jártas *frontend* fejlesztő, szerettem volna gyorsan és hatékonyan egy szerény, azonban *adaptív*⁶ oldalt létrehozni. Ehhez a *Bootstrap*⁷-et alkalmaztam, ami egy ingyenes és nyílt forrású *CSS* keretrendszer. Továbbá a filmekhez, a borítók gyanánt az *Unsplash* API-át⁸ használtam. minden egyéb használt csomag megtalálható a frontend szolgáltatás *package.json* fájljában felsorolva.

1.4. Szervert oldali technológiák

1.4.1. Node.js

A *Node.js*⁹ egy aszinkron eseményvezérelt (event-driven) JavaScript futtatókörnyezet, amely skálázható hálózati alkalmazások fejlesztéséhez lett tervezve.[3]

Noha a *Node.js* nagyon elterjedt és *kiválóan alkalmazható webszerverek, vagy más webes szolgáltatások készítésére*, ugyanakkor megvannak a maga korlátai. Ilyen korlát például az, hogy *egy folyamatban* (process) fut, viszont *nem hoz létre új szálakat minden híváshoz*.[18] Ez annyit jelent a gyakorlatban, hogy ha hosszú, komplikált, nagy számítási kapacitást igénylő műveleteket szeretnél végrehajtani *szinkron* módon, az bizony blokkolja a program futását. Például ha az ajánlórendszer modelljét tanítjuk, vagy a modell segítségével készítünk Top-N ajánlási listát az összes felhasználó számára, azok a felhasználók, akik csak le szeretnék kérni, hogy milyen új filmek vannak, bizony várniuk kell, míg a program végre nem hajtja a műveletet (kész nincs az ajánlási lista), és csak azután tudjuk egyetlen felhasználó kérését fogadni. Ezt orvosolva, a *Node.js*, pontosabban a *libuv* könyvtár¹⁰ létrehozott egy úgynevezett *Event Loop*-ot¹¹, ami egy eseményvezérelt, nem blokkoló I/O modell, mely lehetővé teszi a *Node.js* számára *hogy egyetlen szálon képes legyen kezelni sok aszinkron műveletet*. Tehát, ha a szerver éppen egy felhasználó kiszolgálásával van elfoglalva, egy másik nyugodtan tud kéréseket intézni a szerverhez, és amint lehetséges válaszolni fog rá.

⁶ A responsive és reactive web design-ról bővebben: <https://www.linkedin.com/pulse/reactive-vs-responsive-web-design-complete-comparison-g9fef/>

⁷ Bootstrap hivatalos oldala: <https://getbootstrap.com/>

⁸ Unsplash API - Get a random photo: <https://unsplash.com/documentation#get-a-random-photo>

⁹ Node.js hivatalos oldala: <https://nodejs.org/en>

¹⁰ libuv hivatalos oldala: <https://libuv.org/>

¹¹ Event Loop: <https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick>

1.4.2. Express

Az *Express.js*¹² vagy Express egy Node.js webalkalmazás-keretrendszer, ami széleskörű funkcionálitást kínál webes és mobilalkalmazások számára.[4]

Az *Express* a *felelősségi lánc* (chain of responsibility) viselkedési mintát alkalmazva, köztes szoftverekből (middleware) áll. A felelősségi lánc viselkedési minta lehetővé teszi, hogy kezelők láncolatán kéréseket továbbítson. A kérés beérkezésekor minden kezelő eldönti, hogy feldolgozza-e a kérést, vagy továbbítja azt a következő kezelőnek a láncban. minden kezelő képes megállítani a folyamatot, valamint módosítani az objektumot, melyet átadnak egymásnak.[5] Ezzel lehetővé válnak különböző funkcionálitások, mint például az autentikáció.

1.4.3. Sequelize

A *Sequelize*¹³ egy modern *TypeScript* és *Node.js* objektum-relációs leképzést megvalósító csomag (ORM), amit különböző relációs adatbázisokhoz, mint például *PostgreSQL* vagy *MySQL* használhatunk.[6]

Minden egyéb használt csomag megtalálható a backend szolgáltatás *package.json* fájlban felsorolva.

1.5. Intelligens rendszerben használt technológiák

1.5.1. Python

A *Python*¹⁴ egy interpretált, magas szintű, multiparadigmás (többek közt funkcionális és objektum-orientált) programozási nyelv, dinamikus szemantikával.

Népszerű és széles körben alkalmazzák az *adattudomány* (data science), mesterséges intelligencia (artificial intelligence vagy AI) és a gépi tanulás (machine learning vagy ML) területein is. Ezáltal rengeteg használható eszköz áll rendelkezésünkre, úgy mint a *Surprise* könyvtár, *pandas* vagy a *numpy*. Kiválóan lehet vele szkripteket írni, ahogy az például a 4.2.2. fejezetben is olvasható. Valamint a *Flask*-hoz hasonló könyvtárakkal *API*-okat is lehet velük létrehozni.

1.5.2. Flask

A *Flask*¹⁵ egy *Python*-hoz készült *micro könyvtár*[7], mely segítségével könnyen és egyszerűen tudunk létrehozni egy *REST API*-t, hogy az elkészített ajánlások elérhetők

¹² Express.js hivatalos oldala: <https://expressjs.com/>

¹³ Sequelize hivatalos oldala: <https://sequelize.org/>

¹⁴ Python hivatalos oldala: <https://www.python.org/>

¹⁵ Flask hivatalos oldala: <https://flask.palletsprojects.com/en/3.0.x/>

legyenek a backend részére.

1.5.3. Surprise könyvtár

A *Surprise*¹⁶ könyvtár egy Python *scikit*, mellyel explicit értékeléseket használva ajánlórendszeret tudunk építeni és analizálni.[8]

Megaláthatók benne előre implementált csomagok (package) és modulok (module), úgy mint:

- *prediction_algorithms package*: Különböző előrejelző algoritmusok ajánlások készítéséhez.
- *model_selection package*: Különböző keresztszűréstől validációig mindenhangoláshoz szükséges eszközök.
- *similarities module*: Elemek közötti hasonlóság kiszámítására szolgáló algoritmusok.
- *accuracy module*: Pontossági mérőszámok kiszámításához funkciók.
- *dataset module*: Adatkészletek menedzseléséhez.
- *dump module*: Betanított modellek és elkészült ajánlások szerializálásához, mentéséhez, betöltéséhez.

[8]

Minden egyéb használt csomagot megtalálni az intelligens szolgáltatás *requirements.txt* fájljában, köztük a *pandas* és *numpy* csomagokat.

1.6. Adatbázis-kezelő rendszer

Az *adatbázis* elektronikusan tárolt és elérhető adatok szervezett gyűjteménye. Az *adatbázis-kezelő rendszer* (Database Management System vagy DBMS) az a szoftver, amely az adatok rögzítése és elemzése érdekében kapcsolatban áll a végfelhasználókkal, az alkalmazásokkal és magával az adatbázissal. A DBMS-szoftver emellett magában foglalja az adatbázis kezeléséhez biztosított alapvető eszközöket is. Számos népszerű DBMS létezik, melyeket különböző típusokra lehet osztani az alapján, hogy milyen formában tárolják az adatokat.[11]

¹⁶ Surprise hivatalos oldala: <https://surpriselib.com/>

1.6.1. PostgreSQL

Az alkalmazáshoz *PostgreSQL*¹⁷ (vagy Postgres) választottam. A Postgres egy ingyenes, nyílt forráskódú *relációs adatbázis-kezelő rendszer* (RDMS), amit sokan kedvelnek számos előnye miatt, például támogatja az SQL nyelvet és olyan adattípusokat mint *JSON*, jól skálázható a particionálásnak köszönhetően, biztonságos, stb.[12]

1.7. Docker

A *Docker*¹⁸ egy szoftver platform, melyet alkalmazások fejlesztésére, futtatására és szállítására fejlesztettek. A Docker segítségével elkülöníthető az applikációnk és az infrastruktúra, lehetővé téve a gyorsabb fejlesztés.[14] Operációs rendszer szintű virtualizációt, úgynevezett konténerizációt hajt végre. A hagyományos virtualizációtól ellentétben, a Docker egyetlen operációs rendszer kernelt használ az összes konténer számára, így jelentős mértékben csökkentve a szükséges erőforrásokat és növelte a hatékonyságot. Docker *konténerek* futtatására szolgál, melyek egymástól elszeparált, zárt környezetek. Ezek képesek más konténerekkel és a külvilággal kommunikálni. Az egyes konténerek létrehozásához *képfájlokat* (imagefile vagy image) használ, ezek a konténert leíró fájlok, tartalmaznak minden, ami egy szoftver futtatásához szükséges, beleértve a programkódot, futtatókörnyezetet, könyvtárakat, környezeti változókat és konfigurációs fájlokat. A konténerek segítségével képesek vagyunk automatizálni az applikációk telepítését és futtatását, így azok zökkenőmentesen tudnak különböző környezetekben, izoláltan működni.[13, 15, 16]

1.7.1. Docker Compose

A *Docker Compose* egy eszköz, mely leírja és futtatja a több konténerből álló alkalmazásokat. Leegyszerűsíti az alkalmazás irányítását, segítségével könnyen lehet szolgáltatásokat, hálózatot és köteteket (volume)¹⁹ menedzselni egy egyszerű, átfogó *YAML*²⁰ fájlban.[17]

¹⁷ PostgreSQL hivatalos oldala: <https://www.postgresql.org/>

¹⁸ Docker hivatalos oldala: <https://www.docker.com/>

¹⁹ Docker volumes: lehetővé teszi az adatok tárolását és azok konténerek közötti megosztását.

²⁰ A YAML egy elterjedt, emberek által könnyen olvasható, serializációs nyelv, melyeket gyakran használnak konfigurációs fájlokban. Forrás: <https://en.wikipedia.org/wiki/YAML>

2. fejezet

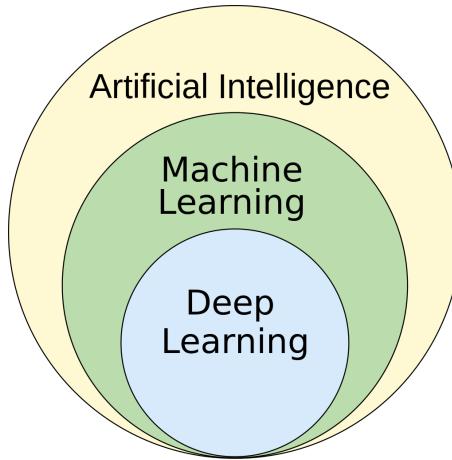
Gépi Tanulás

2.1. Mi az a Gépi Tanulás?

A gépi tanulás (Machine Learning) kifejezést 1959-ben *Arthur Samuel*, az IBM egyik alkalmazottja alkotta meg. A gépi tanulásra úgy is hivatkoztak, hogy *öntanító* vagy *önoktató számítógépek* („self-teaching computers”).[19]

A gépi tanulás a számítógép-tudomány és *mesterséges intelligencia* (MI, angolul artificial intelligence vagy AI) egy ága, ami lehetővé teszi a számítógépek olyan dolgok elvégzésére, melyre nem programozták őket – nem adtak utasításokat, nem definiálták előre a lépéseket. Ezt nem úgy kell érteni, hogy a gépek programozás nélkül teljesítenek feladatokat, sokkal inkább úgy, hogy a beviteli adatokat (input data), nem pedig a beviteli parancsokat (input command) használva hajtják azt végre.[20]

Fontos észben tartani, ahogy az a 2.1. ábrán is látható, a gépi tanulás nem ekvivalens a mesterséges intelligenciával, hanem csak egy részhalmaza annak. Míg a *gépi tanulás* olyan algoritmusokat jelent, melyek képesek az említett tanulásra, előrejelzéseket és döntéseket elvégezni a beviteli adat alapján, addig a *mesterséges intelligencia* alatt a program vagy mesterségesen létrehozott tudat által megnyilvánuló intelligenciát értjük.[21, 22] *Alan Turing* volt az első, aki jelentős kutatásokat végzett a mesterséges intelligencia területén, amit *gépi intelligenciának* (machine intelligence) nevezett.[21, 22]



2.1. ábra. A mesterséges intelligencia részhalmazai¹

A gépi tanulást megannyi területen használják, mint például:

- csalások feltárása (fraud detection)
- levélszemét szűrők (spam)
- beszéd felismerés (speech recognition)
- természetes nyelvek feldolgozása (natural language processing vagy NLP)
- gépi látás (computer vision)
- egészségügy
- önvezető járművek
- ajánló rendszerek (recommender systems) melyet a szakdolgozatom témajának választottam, bővebben a 3. fejezetben olvashatunk róla.

2.2. A gépi tanulás lépései

A [23] felhasználva a gépi tanulást a következő lépésekre lehet bontani:

1. **A probléma definiálása:** Mit szeretnénk megoldani?
2. **Adatgyűjtés:** Az algoritmusunk tanításához.
3. **Adatok előkészítése, előfeldolgozása:** Például a hiányzó értékek kezelése, normalizálás.

¹ Forrás: [27]

4. **Az adatok elemzése:** Annak érdekében, hogy megértsük a mintákat, trendeket és összefüggéseket. Ezek segítenek a feladatokhoz szükséges jellemzőkészletek definiálásában (feature engineering).
5. **Jellemzők definíálása** (Feature engineering): „A feature engineering az adatbányászat első fázisa, amelyikben az adatokat gyűjtjük össze, egészen pontosan az általunk fontosnak tartott “jellemzőket” (features).”[24]
6. **Algoritmus kiválasztása:** Az eddig végzett lépések alapján a megfelelő ML algoritmus kiválasztása. Fontos szempontok pl. a probléma, amit meg szeretnénk oldani és az adat természete.
7. **Az adat elosztása:** A gépi tanuláshoz a teljes adathalmazt általában két vagy három diszjunkt részre bontjuk, lásd a 2.3.3. fejezetben.
8. **A modell tanítása:** „A tanító adatbázis alapján, a kiértékelési metrika, mint célfüggvényre optimalizálva, a gépi tanulás során felépítünk egy modellt. A modell a döntési szabály, ami tartalmazza az egyedek jellemzői közti feltárt kapcsolatokat /mintázatokat/szabályszerűségeket.”[26]
9. **A modell kiértékelése:** Különböző metrikákkal (bővebben a 3.6. fejezetben) (például RMSE) a tesztkészletet felhasználva.
10. **Hiperparaméter-finomhangolás** (Hyperparameter Tuning): A modell paramétereinek finomhangolása a jobb eredmény eléréséhez
11. **Üzembe helyezés**
12. **Felhasználói visszacsatolás** (Feedback Loop): A rendszer javításának, fejlesztésének érdekében.

2.3. Adat

Ahogy az előző pontban is látható, a beviteli adat, amivel majd a modell tanítva lesz, nagyban befolyásolja az algoritmus kimenetét. A több mennyiségű adat nem feltétlenül jelent azt, hogy jobban teljesít majd a betanított modell. Nagyon fontos, hogy az adatot megfelelően előkészítsük a feldolgozásra. Kulcsfontosságú, hogy lehetőleg csak a releváns adatokat tartsuk meg, ugyanakkor, ahogy a későbbiekben olvasható, előfordulhat, hogy a modell új, számunkra ismeretlen összefüggéseket fedez fel (lásd a 2.4.2. fejezet). Megeshet, hogy az adatban sok jellemző hiányos (missing data), vagy nulla értékű (sparse data).

- **Ritka adat**: olyan adatkészletekre utal, ahol sok bejegyzés értéke 0 vagy egy alapértelmezett nem informatív érték (például null).
- **Hiányos adat**: olyan adathiányra utal, mely valamilyen hiba miatt, az adatok váratlanul nem állnak rendelkezésünkre.

Ritka adat				
Dátum	Szenzor 1	Szenzor 2	Szenzor 3	Szenzor 4
2024.01.01.	0,12	0,00	0,00	0,53
2024.01.02.	0,25	0,00	0,00	0,87
2024.01.03.	0,12	0,00	0,77	0,39
2024.01.04.	0,64	0,00	0,00	0,12
2024.01.05.	0,21	0,00	1,18	0,61

Hiányzó adat				
Dátum	Szenzor 1	Szenzor 2	Szenzor 3	Szenzor 4
2024.01.01.	0,12	???	0,26	0,53
2024.01.02.	0,25	???	0,33	0,87
2024.01.03.	0,12	???	0,77	0,39
2024.01.04.	0,64	???	0,89	0,12
2024.01.05.	0,21	???	1,18	0,61

2.2. ábra. Ritka és hiányzó adatok reprezentálása²

A 2.2 ábrán látható, hogy míg a *ritka adat* az a *Szenzor 2-nél* és sok helyen a *Szenzor 3-nál* 0 értékű – tehát a mérések és az adat küldések megtörténtek – addig a *hiányzó adat* táblában a *Szenzor 2* valamiért nem küldött mérési adatokat. Ugyanakkor gépi tanulás kontextusában mindenkorre gyakran csak „sparse data”-ként hivatkoznak. Vannak olyan módszerek, algoritmusok, melyek nem működnek jól ilyen adatokkal (például memória-alapú kollaboratív szűrők, lásd a 3.4.1. fejezetben), de vannak, amelyeket ilyen esetekre használnak (például *SVD*, lásd a 3.4.3. fejezet).

Az adatok sok forrásból jöhetnek, *különböző formákban*. Ilyen szempontból megkülönböztethetünk:

- **Struktúrált adat**: Adatbázis táblák.
- **Strukturálatlan adat**: Szöveg, kép, hang, videó, stb.
- **Félig Struktúrált (Semi-Structured) adat**.

Valamint az *adat mennyisége* is eltérő lehet:

- **Data**: Általában struktúrált formában van tárolva, pár terrabájt nagyságrendig.

² Forrás: <https://stats.stackexchange.com/questions/267322/difference-between-missing-data-and-sparse-data-in-machine-learning-algorithms>

- **Big Data:** Strukturálatlan vagy félre strukturált formában van tárolva, pár terabájttól egészen több exabájt ($1EB \approx 10^6TB$) nagyságrendig.

Ezek persze több dologban is eltérhetnek, ugyanakkor érdemesnek tartottam megemlíteni ezeket a szempontokat.

A *táblázatos adat* (tabular-data) olyan adatkészlet, amely táblákban, sorokba és oszlopokba van rendezve.[28] Az oszlopok a *jellemzőket* (variable, dimension, attribute, feature) és a sorok reprezentálják az *egyedeket* (példány, instance, row, case, value), egy adott jellemzőn való megfigyeléseket (lásd 2.3. ábra).[20, 26]

Jellemző (feature)				
Film Címe	Akció	Komédia	Fantasy	Sci-Fi
Star Wars: A klónok háborúja	4	1	4	5
Gyűrük Ura: A két torony	4	1	5	1
Coco	1	3	5	1
Így jártam anyátokkal	1	5	1	1
A pláza ásza	3	5	1	1

2.3. ábra. Táblázatos adat[26]

Az oszlopokat lehet *vektoroknak* is nevezni és több vektorra lehet *mátrixként* is hivatkozni (lásd 2.4. ábra). A felügyelt tanulásnál, a sorok már léteznek az adatkészletben és arra használjuk őket, hogy a mintázatokat fedezzük fel az egymástól független jellemzők közt.[20]

Mátrix				
Film Címe	Akció	Komédia	Fantasy	Sci-Fi
Star Wars: A klónok háborúja	4	1	4	5
Gyűrük Ura: A két torony	4	1	5	1
Coco	1	3	5	1
Így jártam anyátokkal	1	5	1	1
A pláza ásza	3	5	1	1

2.4. ábra. Vektor és Mátrix táblázatos adatnál[20]

2.3.1. Az adat tisztítása

Az adat változatossága miatt – például hiányzó, ritka, vagy éppen strukturálatlan adat – az adatot elő kell készíteni mielőtt a modellt elkezdjük rajta tanítani. Ezt a lépést angolul „Data Scrubbing”-nak nevezik.

A tisztítás alatt több dologra is oda kell figyelnünk, úgy mint:

- Duplikációk
- Szenzitív információk törlése vagy anonimizálása
- Ritka, vagy hiányos adat
- Az adat standardizálása
- Normalizáció

2.3.2. Az adat gyűjtése

Az adat gyűjtése történhet *explicit* és *implicit* módon.

Az explicit adatgyűjtés általában a rendszer által, a felhasználótól kért adatbevitelt jelent, például:

- Adjon értékelést
- Válassza ki a listából az általa legjobban preferált cikkeket

Mivel a felhasználók nagy többsége valamely oknál fogva nem szeret explicit értékelést adni, ezért sokszor az implicit adatgyűjtés jelenti a megoldást. Ez a módszer számos előnyvel rendelkezik, többek közt lehetővé teszi sokkal szélesebb körű és változatosabb adatok gyűjtését:

- a felhasználó mikor és mivel lépett interakcióba (például megtekintett egy árucikket)
- egy filmet, videót vagy terméket mennyi ideig nézett
- mit és mivel vásárolt együtt

2.3.3. Az adat készletekre való felosztása

A gépi tanuláshoz az a teljes adathalmazt általában három diszjunkt részre bontjuk:

- **Tanító készlet** (train set): A rendelkezésre álló adat nagy részét – az összes adat 60-80%-át – a modell tanítására, azaz a modell paramétereinek a beállítására használjuk fel.
- **Validációs készlet** (validation set): Ezt az adathalmazt a *hiperparaméterek finomhangolására* (lásd 5.2. fejezet), valamint *regularizációra* (az *idő előtti leállítás* segítségével) használjuk, amellyel elkerülhető a túlilleszkedés. Az összes adat 10-20%-át kitevő részhalmaz.

- **Tesztelési készlet** (test set): A tesztelési (esetleg tesztelő vagy teszt) készlet egy a modell számára új, ismeretlen adathalmaz. Ezt használjuk a modell teljesítményének mérésére (lásd 3.6. fejezet). Az összes adat 10-20%-a.

[25]

Ugyanakkor előfordul, hogy az adatot csak *tanító* és *tesztelő* készletekre bontják. Például akkor, ha nincs elegendő adatmennyiség vagy számítási kapacitás, vagy – ahogy a szakdolgozatomban is – amikor *keresztszabvádációt* (lásd 3.5.3. fejezet) használnak, mely lehetővé teszi az adat hatékonyabb felosztását.

2.4. Tanulási típusok

2.4.1. Felügyelt tanulás

A felügyelt tanulás (supervised learning) során a gépet *címkezett* (labeled) adatkészletekkel tanítják, hogy felismerje az adatok közötti mintákat és mintázatokat, majd megtanulja a bemutatott példákat megfelelően címkezni. Mivel a tanítási adatokat már előre ellátták a helyes címkekkel, a modell visszajelzést kap a döntéseinek pontosságáról. Ennek köszönhetően a betanított modell képes lesz arra is, hogy olyan új adatokat is címkekkel ellásson, amelyekkel korábban még nem találkozott.[29, 30]

A felügyelt tanulást két fő kategóriába soroljuk: **klasszifikációs** (vagy osztályozó) és **regressziós** algoritmusokra. A klasszifikációs feladatok során gyakran alkalmazott módszer a *Logisztikus regresszió*, míg a regressziós problémák megoldására gyakran használják a *Lineáris regressziót*.[19, 29]

2.4.2. Felügyelet nélküli tanulás

A felügyelet nélküli tanulásnál (unsupervised learning vagy self-learning) az emberek által nem ellenőrzött, címkezetlen adatokat szolgáltatnak bemenetként. Mivel ezeket az adatokat nem látták el címkekkel, így a gépnek lehetősége van arra, hogy saját maga fedezze fel az adatok közötti összefüggéseket és kapcsolatokat.[31]

A felügyelet nélküli tanulás különösen jól működik nagyobb komplexitású feladatok esetében, mint például nagy adatkészletek klaszterezésénél – „*A klaszteranalízis egy olyan dimenziócsökkenő eljárás, amellyel adattömböt tudunk homogén csoportokba sorolni, klasszifikálni.*”[32].

2.4.3. Félig felügyelt tanulás

A félig felügyelt tanulásnál (semi-supervised learning) az adatok diverzitása, struktúrálatlansága és nagy volumene, valamint az emberi kapacitás korlátozottsága miatt az

adatok többsége nem rendelkezik címkékkal. Azonban, egy kis mennyiséggű címkézett adat felhasználásával jelentősen javítható a tanulási folyamat sebessége és pontossága. Ezzel együtt járó hátránya, hogy a rendszer a címkézett adatok esetleges hibáit is megtanulhatja.[33, 34]

2.4.4. Megerősítéses tanulás

A *megerősítéses tanulás* (reinforcement learning) a modelleket úgy tanítja döntések hozatalára, hogy az optimálishez közeli végeredményt érjék el. A megerősítéses tanulási algoritmusok a *jutalmazási* és *büntetési mechanizmusokra épülnek*, és a kapott vissza-jelzésből vonnak le következtetéseket, annak érdekében, hogy saját maguk fedezzék fel a cél elérésének optimális módját. Képesek a késleltetett jutalmazás alkalmazására is, ami lehetővé teszi számukra, hogy hosszú távon a legjobb eredményeket érjék el, még akkor is, ha ez rövid távon nem tűnik a legkifizetődőbbnek.[35]

A való életben ez nagyban hasonlít ahhoz, ahogy az emberek, vagy például a kutyák tanulnak. Vegyük azt az esetet, amikor egy kutyát arra tanítunk, hogy az „ül” parancsra leüljön. Jutalomfalatokat használhatunk, és amikor a kutyának mondjuk a parancsszót és ő leül, kap egy falatot; ha viszont nem tesz eleget a parancsnak, akkor nem jutalmazzuk. Így a kutya idővel megtanulja a parancsszót és a hozzá tartozó cselekvést, valamint azt, hogy hogyan szerezheti meg a lehető leggyorsabban a kívánt jutalmat.

2.5. Hiperparaméter optimalizálás

Hiperparaméterek azok a konfigurációs beállítások, amelyek meghatározzák a gépi tanulási modellek működését, de nem keverendők össze sem az adathalmaz adatalemeivel, sem a modellek belső paramétereivel, mint például a súlyok vagy a bias.

Ezek a paraméterek, mint például a neurális hálózatokban a *rétegek száma*, a *tanulási sebesség*, vagy az *aktivációs funkciók*, nem a tanulási folyamat során, hanem előzetesen, manuális beállítással vagy automatizált keresési módszerekkel kerülnek optimális értékre állításra.

A hiperparaméter-finomhangolás, amelyet az angol szakirodalom „Hyperparameter-Tuning” vagy „Hyperparameter-Optimization” néven ismer, elengedhetetlen a modell hatékonyságának maximális kiaknázása szempontjából. Ennek ellenére ez a feladat gyakran jelentős kihívást jelent a számos lehetséges konfigurációs kombináció miatt. Az optimális hiperparaméter-kombinációk megtalálásának több módja létezik, de a legegyszerűbbek közé tartozik a *rácskeresés* (grid search) – melyet az alkalmazásban használok, és az 5.2. fejezetben be is mutatom – vagy a *paraméter-söprés* (parameter sweep), amely egy kimerítő keresési technika a hiperparaméter-tér kézzel meghatározott

részhalmazában. Ezt a keresési módszert valamilyen teljesítménymutatóval kell vezérelni, mint például a keresztvalidáció (cross-validation) (lásd a 3.5.3. fejezet).[36, 37]

3. fejezet

Ajánló rendszerek

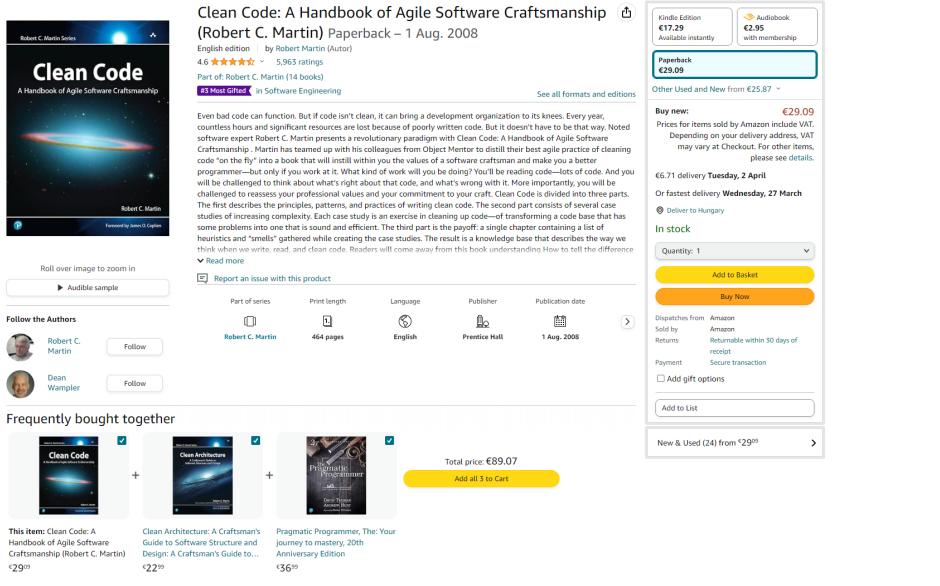
Az *ajánló rendszerek* (recommender-, recommendation system, engine vagy algorithm) speciális *információ szűrő rendszerek*, amelyek felhasználói és termékprofilokat építenek tanuló algoritmusok segítségével, majd a modellek alapján ajánlanak olyan *tartalmakat* (film, zene, videó, stb.) vagy *termékeket* a felhasználónak, amely nagy valószínűsséggel érdekes lesz számára.[38, 39, 40]

3.1. Mely problémákra nyújtanak megoldást ?

Az internet növekedése, az online tartalom mennyisége miatt a felhasználók számára nehézzé vált az információ szűrése, a releváns tartalmak megtalálása, ez az *információs túlterhelés*.[40]

Ugyanakkor, üzleti szempontból is megvannak a nehézségei az online térben való érvényesülésnek, hiszen a konkurenciák száma is nőttön nő, akiknek a kínálata is sokkal széleskörűbb lehet. Így, hogy minél több és minél elégedettebb felhasználó legyen – és természetesen minél többet fogyasszanak – az üzleteknek érdekük segíteni a felhasználót, hogy könnyen megtalálják a számukra megfelelő termékeket, esetleg olyan termékeket is megvásároljanak, amit amúgy nem állt szándékukban, viszont látják, hogy mások, a termék mellé, mely más termékeket vásárolták (Lásd 3.1 ábra alján a „Frequently bought together” részt).

Ahogy azt olvashatjuk a 3.4 részben, erre és ehhez hasonló problémákra különböző megközelítéseket hoztak létre.



3.1. ábra. Példa a gyakran együtt vásárolt termékekre¹

3.2. Ajánlási feladat formális leírása

Ebben az alfejezetben az ajánlási feladat formális leírásához a [40] munkájában leírtakat használom.

3.2.1. Hasznosságfüggvény

Ha C jelöli a felhasználókat és S a termékeket, akkor

$$u: C \times S \rightarrow \mathbb{R}, \text{ ahol } \mathbb{R} \text{ halmaz teljesen rendezett,}$$

jelöli a *hasznosságfüggvényt*.[40]

3.2.2. Tanulási feladat

Hatórozzuk meg azt az $u': C \times S \rightarrow \mathbb{R}$ leképzést, amely a lehető legjobban közelíti u -t és teljesen definiált a teljes $C \times S$ téren.[40]

3.2.3. Ajánló rendszer kimenete

$\forall c \in C$ vásárló számára a legérdekesebb/leghasznosabb új termék(ek) megadása, azaz:

$$s_c \in S, \text{ ahol } u(c, s_c) \text{ maximális.}$$

¹ Forrás: <https://amazon.de>

”[40]

3.3. Hasonlósági függvények

Ahogy azt a későbbiekben is olvashatjuk, az ajánlórendszerek számára szükséges, hogy meg tudjuk határozni a felhasználók vagy az elemek közötti *hasonlóságot*. Erre a célra különböző módszerek állnak a rendelkezésünkre, például:

- **Euklideszi norma** (Euclidean distance): Két pont közé eső szakasz hosszának meghatározására szolgál.[52]

$$\sqrt{\sum_{i=1}^n |x_i - y_i|^2}$$

- **Pearson-féle korrelációs együttható** (Pearson correlation coefficient): „olyan mérőszám, amely az erősségét és az irányát mutatja meg egy lineáris kapcsolat két változója között, amely a változók kovarianciája osztva a standard szórás. Ez a korrelációs együttható legismertebb és leggyakrabban használt típusa.”[53]
- **Spearman rangkorrelációs együttható** (Spearman’s rank correlation coefficient): „azt jelzi, hogy a két változó közötti kapcsolatot hogyan lehet monoton függvényel leírni.”[53]
- **Jaccard-hasonlóság** (Jaccard similarity): A halmazok közötti hasonlóságot úgy méri, hogy a halmazok metszetének méretét elosztja a halmazok uniójának méretével. Különösen hasznos azoknál a rendszereknél, ahol az adatok binárisak, például megtekintette/nem tekintette meg.[44]

3.3.1. Koszinusz hasonlósági függvény

A *koszinusz hasonlóság* (cosine similarity) a dokumentumok hasonlóságának egyik legelterjedtebb mértéke, mely két nem nulla vektor hasonlóságának megállapítására szolgál. A koszinusz hasonlóságnál, ahogy a nevéből is következtethető, a *vektorok által bezárt szög koszinusza*, vagyis a vektorok *skaláris szorzata*² osztva a hosszuk szorzatával.[44, 49]

3.1. Definíció. Két nem nulla vektor koszinusza az *Euklideszi skalárszorzat* képletének segítségével számítható, ahol x és y két nem nulla vektor, és az általuk bezárt θ szög koszinusza meghatározható:

$$\cos(\theta) = \frac{x \cdot y}{\|x\| \|y\|}$$

² Skaláris szorzatról bővebben: [50]

Ahol $x \cdot y$ az x és y vektorok közötti *skaláris szorzatot*, n pedig a dimenziók számát jelöli:

$$x \cdot y = \sum_{i=1}^n x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

$\|x\|$ és $\|y\|$ pedig x és y vektorok hosszát, vagy Euklideszi normáját jelenti:

$$\|x\| = \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

és

$$\|y\| = \sqrt{\sum_{i=1}^n y_i^2} = \sqrt{y_1^2 + y_2^2 + \cdots + y_n^2}$$

[44, 50, 51]

Példa

Vegyük például a *Free Guy* (2021) akció/vígjáték és a klasszikus *Drágán add az életed!* (1988) akció/thriller filmeket. Az egyszerűség kedvéért, nézzük csak az akció és a vígjáték műfajokat. Képzeljünk el egy koordináta rendszert, ahol az x az akció az y tengely pedig a vígjáték jellemzőt reprezentálja egy 0-1 skálán. Ha ezen elhelyezzük a *Drágán add az életed!* filmet az a [1,0] míg a *Free Guy* az [1,1] pontokon helyezkednek el. Ha az origóból húzunk egy-egy vektort ezekbe a pontokba, megkaphatjuk az általuk bezárt θ szöget, és így kiszámíthajuk a köztük lévő hasonlóságot a *koszinusz hasonlósági* függvényt használva.

Többdimenziós terek

Az előző példában ugyan az egyszerűség kedvéért két dimenzióban szemléltettem a koszinusz hasonlósági függvényt, ugyanakkor ettől több is lehet. Az applikációban használt adatkészletben számszerűen 19 darab műfaj található. A fenti definíció azonban ugyanúgy alkalmazható 19 dimenziós vektorokra, mint 2 dimenziósokra.

3.4. Különböző megközelítések

Az ajánló rendszerek algoritmusainak különféle megközelítése ismert:

- Kollaboratív szűrők (lásd a 3.4.1. fejezetben)
- Tartalomalapú módszerek (content-based filtering): a felhasználó által kedvelt jellemzőkkel bíró termékek (lásd a 3.4.2. fejezet)

- Mátrix faktorizáció (lásd a 3.4.3)
- Mély tanulás (deep learning)
- Faktorizáló gépek (factorization machines)
- Munkamenet-alapú ajánlások (session-based recommendations)
- Gráf-alapú ajánlások (graph based recommendations)
- Hibrid modellek (hybrid models): A különböző megközelítéseket kombinálva, azok előnyeit egyesítve, hátrányaikat ellensúlyozza.

3.4.1. Kollaboratív szűrés

A *kollaboratív szűrés* (collaborative filtering) *múltbeli* felhasználói interakciók és preferenciák, implicit vagy explicit értékelések alapján ajánl új, még nem értékelt termékeket. Azt a feltételezést veszi alapjául, hogy ha az embereknek megegyezett az ízlése a múltban (például többen is ugyan azokat a filmeket kedvelték, vagy éppen ellenkezőleg) akkor a jövőben is egyetértésre jutnak.[39, 40, 41]

Felhasználó	Star Wars: A klónok háborúja	Gyűrűk Ura: A két torony	Coco	Így jártam anyátokkal	A pláza ásza
Felhasználó 1	5	5	3	?	2
Felhasználó 2	3	2	4	5	5
Felhasználó 3	4	3	2	4	4
Felhasználó 4	5	?	2	2	3

Felhasználó	Star Wars: A klónok háborúja	Gyűrűk Ura: A két torony	Coco	Így jártam anyátokkal	A pláza ásza
Felhasználó 1	5	5	3	2	2
Felhasználó 2	3	2	4	5	5
Felhasználó 3	4	3	2	4	4
Felhasználó 4	5	5	2	2	3

3.2. ábra. Kollaboratív szűrés reprezentálása

Ahogy a 3.2. ábrán, a felső táblázat mutatja, a *Felhasználó 1* és a *Felhasználó 4* sor hiányos elemekkel rendelkeznek. Mivel a *Felhasználó 1* kedveli a *Star Wars-t* és a *Gyűrűk Urat*, viszont a *Pláza ászát* nem, ezért logikusnak tűnik, hogy a sci-fi és a fantasy műfajokat szereti, a komédiát viszont nem. Párhuzam vonható közte és a *Felhasználó 4* preferenciái közt, ezért következtethetjük, hogy a *Felhasználó 4*-nek tetszeni fog a *Gyűrűk Ura*, míg a *Felhasználó 1*-et nem fogja érdekelni az *Így jártam anyátokkal*. Nagyjából ez a gondolatmenet alkalmazható a gépi tanulás, különösen a kollaboratív szűrési algoritmusok területén is. Azonban fontos figyelembe venni, hogy nem felhasználó párokról beszélünk, hanem hasonló ízléssel bíró felhasználók által adott értékelések (legyenek azok implicit vagy explicit értékelések) aggregálásáról.

A [39, 40, 41] források szerint, a kollaboratív szűrési módszereket két osztályra lehet bontani, a *memória alapú* (memory-based) és a *modell alapú* (model-based) megközelítésekre. A *memória alapú* megközelítések közvetlenül az adatbázisban lévő felhasználó-elem interakciókat használják fel az ajánlások előállításához, míg a *modell alapú* megközelítések *előrejelző modelleket* építenek a felhasználói preferenciák és az elemek közötti kapcsolatok modellezésére.

Memória alapú megközelítések

A *memória alapú megközelítés* további csoportokra osztható:

- **Felhasználó alapú szűrés** (user-based filtering): Felhasználók közötti hasonlóság alapján tesz ajánlásokat (lásd a fentebb lévő példát és a 3.2 ábrát). Általában az alábbi két lépésre bontható az algoritmus:

1. Az aktuális felhasználóval (aki számára készíti az ajánlást) hasonló ízlésű felhasználókat keres az értékelési minták vizsgálatával
2. A hasonló ízlésű felhasználók értékeléseit felhasználva becslést készít az aktuális felhasználó számára

[41]

- **Elem alapú szűrés** (item-based filtering): Az elemek közötti hasonlóság alapján tesz ajánlásokat, például ha sok felhasználó szeret egy könyvet, és azok közül soha szeretnek egy másikat, akkor annak, aki kedveli az egyiket, ajánljuk a másikat (lásd a 3.1 ábra alját). Általában az alábbi két lépésre bontható az algoritmus:

1. Épít egy elem-elem mátrixot amely meghatározza az elempárok közötti kapcsolatot
2. A mátrix vizsgálatával és a felhasználó adatainak egyeztetésével következtet az aktuális felhasználó ízlésére.

[41]

- **Hibrid** (hybrid): A felhasználó és elem alapú módszerek előnyeit kombinálja.
[43]

A felhasználó- és az elem-alapú szűrésekre gyakran hivatkoznak *szomszéd-alapú* kollaboratív szűrként (neighborhood-based collaborative filtering) is.

Különböző algoritmusokat használnak a hasonlóságok mérésére, például *k-legközelebbi szomszéd* (k-nearest neighbor, röviden k-NN) és a *Pearson korreláció* (Pearson Correlation) (lásd 3.3. fejezet).[39, 41]

Modell alapú megközelítések

A *modell alapú* megközelítésekre példa:

- **Bayes-hálók** (Bayesian networks): A csomópontok a termékek, és lehetséges általapotaik az egyes értékelések.[40] Ezek a hálózatok a valószínűségi következtetések és döntéshozatali folyamatok modellezésére szolgálnak.
- **Klaszterezésen alapuló módszerek** (clustering models): A hasonló ízlésű felhasználókat egy-egy osztályba soroljuk, lehetővé téve az ajánlások személyre szabását a felhasználók klaszterei alapján.[40]
- **Mátrix faktorizáció** (matrix factorization): A felhasználó-elem interakció mátrixot két kisebb dimenziójú mátrix szorzatára bontják (lásd a 3.3. ábra). Ilyen módszer például az *SVD* (szinguláris érték dekompozíció vagy angolul singular value decomposition) amit az applikációban is implementáltam, bővebben később, a 3.4.3. fejezetben.
- **Neurális hálózatokon alapuló megközelítések** (neural network-based): Az adatokból mély tanulási modellek segítségével történő mintázatfelismerés és ajánlás, ami képes a bonyolultabb felhasználói interakciók és preferenciák modellezésére.[45]

[41]

		Item			
		W	X	Y	Z
User	A		4.5	2.0	
	B	4.0		3.5	
	C		5.0		2.0
	D		3.5	4.0	1.0

=

		W	X	Y	Z
User	A	1.2	0.8		
	B	1.4	0.9		
	C	1.5	1.0		
	D	1.2	0.8		

X

		W	X	Y	Z
Item	W	1.5	1.2	1.0	0.8
	X	1.7	0.6	1.1	0.4
	Y				
	Z				

Rating Matrix User Matrix Item Matrix

3.3. ábra. Mátrix faktorizáció film értékelés táblán³

Előnyei

A kollaboratív szűrés nagy előnye, hogy a gépnek nem kell elemeznie a komplex elemeket, úgymond megértenie, mit is jelent az, hogy film, és milyen tulajdonsággal bírnak, hogyan lehet őket definiálni.[39, 41]

³ Forrás: [79]

Hátrányai

Ugyanakkor megvannak a maga hátrányai is, legfőbbképpen a *hideg indítás* (cold start), *skálázhatoság* (scalability) és a *ritka adat* (sparsity):

- **Hideg indítás:** Új felhasználóknál vagy elemeknél nem áll rendelkezésünkre elég adat hogy pontos ajánlást tudjunk adni.
- **Skálázhatoság:** A nagy mennyiségű adat, a felhasználóbázis és a kínálat miatt nagy számítási kapacitásra van szükség. Ha meg szeretnénk találni a hasonló ízlésű felhasználókat, akkor ki kell számítanunk minden felhasználó párra a hasonlóságot egy függvényel, majd a kapott eredményt rendeznünk kell, hogy megtalálhassuk a leginkább hasonló felhasználókat. Mivel ekkora mennyiségű adat nem fér el a RAM-ban, ezért nagyon lassú lehet ez a folyamat. Erre egy megoldást jelenthet egy gráf létrehozása.
- **Ritka adat:** Noha a kínálat hatalmas, a legtöbb felhasználó ennek csak a töredékével lép interakcióba és annál is kevesebbről adnak explicit értékelést.

[39, 40, 41]

Felhasználó-alapú kollaboratív szűrés formális leírása

A 3.2. fejezetben használt jelöléseket használva – C a felhasználók, S a termékek halma, $c \in C$, valamint $s \in S$.

Egy termék hasznossága ♦ egy adott skálán mozogó értékelés jelöljük:

$$r_{c,s} \equiv u(c, s)$$

Ez azt mutatja, hogy a felhasználó, milyen magasra pontozta, vagy értékelte (volna) az adott terméket.[40]

Egy adott felhasználó számára, egy még nem értékelt termék értékelésének becslése hasonló felhasználók által adott értékelések összesítéséből ♦

$$\hat{r}_{c,s} \leftarrow \forall r_{c_i,s} \text{ alapján, ahol } c \sim c_i$$

[40]

Az ajánlás menete ♦

1. Felhasználópárosok *hasonlósági faktorának* megállapítása:

$$\forall x, y \in C: sim(x, y) \text{ meghatározása}$$

2. Adott felhasználóhoz hasonló ízlésű felhasználók megkeresése

$$\forall c \in C: c \rightarrow \hat{C} = \{c' | c' \in C, sim(c, c') \text{ nagy}\}$$

3. A hasonló ízlésű felhasználók által adott értékek aggregálásával az ismeretlen értékelés becslése:

$$\hat{r}_{c,s} = aggr_{\forall c' \in \hat{C}}(r_{c',s})$$

[40, 41]

Felhasználó párok hasonlóságának meghatározása ♦ Általában a felhasználók által közösen értékelt S_{xy} termékhalmaz alapján számítjuk:

$$S_{xy} = \{s \in S | r_{x,s} \neq \emptyset, r_{y,s} \neq \emptyset\}$$

A két felhasználó által értékelt termékek pontszámait egy-egy x és y vektorként használva definiálhatjuk a hasonlóságot a két vektor által bezárt szög koszinuszaként (Koszinusz hasonlósági függvény látható a 23. oldalon):

$$sim(x, y) = cos(x, y) = \frac{x \cdot y}{|x| \cdot |y|} = \frac{\sum_{s \in S_{xy}} r_{x,s} \cdot r_{y,s}}{\sqrt{\sum_{s \in S_{xy}} r_{x,s}^2} \cdot \sqrt{\sum_{s \in S_{xy}} r_{y,s}^2}}$$

[40]

Értékek aggregálása és súlyozása ♦ A kollaboratív szűrőknél kulcsfontosságú probléma, hogy az értékeket, preferenciákat hogyan kombináljuk, súlyozzuk.[41]

– Egyszerű átlag

$$\hat{r}_{c,s} = \frac{1}{N} \sum_{\forall c' \in \hat{C}} r_{c',s}, \text{ ahol } N = |\hat{C}|$$

– Felhasználók hasonlóságával súlyozott átlag

$$\hat{r}_{c,s} = k \cdot \sum_{\forall c' \in \hat{C}} sim(c, c') \cdot r_{c',s}, \text{ ahol } k = \frac{1}{\sum_{\forall c' \in \hat{C}} |sim(c, c')|}$$

– Súlyozott átlag, normalizált értékelésekkel

$$\hat{r}_{c,s} = \bar{r}_c + k \cdot \sum_{\forall c' \in \hat{C}} sim(c, c') \cdot (r_{c',s} - \bar{r}_{c'})$$

[40]

3.4.2. Tartalommalapú módszerek

A *tartalommalapú szűrés* (content-based filtering) az elemek jellemzőiből és a felhasználó által adott implicit vagy explicit értékelések, preferenciák alapján generál személyre szabott ajánlásokat, ugyanakkor nem a felhasználók közötti hasonlóságot felhasználva, hanem az *elemek közti jellemzőkre alapozva*.[46, 47]

Ezt a módszert akkor érdemes használni, ha a termékekéről megfelelő minőségű és mennyiségű információ, jellemző (lásd a 2.3. fejezet, 2.3 ábra) található, a felhasználókról viszont nincs elegendő, például új felhasználó (lásd hideg indítás probléma 3.4.1) vagy csak kevés interakcióról van adatunk (lásd ritka adat a 2.3. fejezetben), ami következtében nehezen tudnánk a felhasználó hasonlóságot modellezni.[39]

Megközelítések

A tartalommalapú módszereket is többféleképpen meg lehet közelíteni, például:

- **Jellemző reprezentáció:**
 - **Vektortér modell** (vector space model): Az elem és felhasználói profilk vektorokként vannak reprezentálva a többdimenziós térben, ahol minden dimenzió az elem egy jellemzőjének felel meg.
 - **TF-IDF** (Term Frequency-Inverse Document Frequency): Gyakran használt módszer a szövegalapú elemknél. Ez a megközelítés nagy hangsúlyt fektet a dokumentumon szövegére, szavakra, azok előfordulására.[54]
- **Felhasználó profil modellezés** implicit és explicit visszajelzésekből.
- **Elem profil modellezés:**
 - **Metaadat alapú**: Az elem metaadatait használja, úgy mint a műfaj, szerző, rendező, stb. Jól használható filmknél, könyveknél, egyéb médiatartalmaknál.
 - **Tartalom elemzés** (content analysis):
 - *szöveges tartalmaknál* a természetes nyelvek feldolgozását (natural language processing vagy NLP),
 - *képekhez és videókhoz* pedig a gépi látást (computer vision) használja.
- **Osztályozás és regresszió**: A felhasználó értékelésére vagy annak ízlésére ad becslést az elemek jellemzői alapján.
- **Neurális hálózat** (Neural network) és **Mély tanulás** (Deep Learning): Komplex modelleket készítenek.

- **Hibrid módszerek**

[39]

Előnyei

- Nem a felhasználó adatain, interakción alapul, ezért jól működik például a *hideg indításnál* (lásd 3.4.1. fejezet).
- Független a többi felhasználótól
- Új vagy kevésbé felkapott elemeket is ajánlhat (hosszú farok vagy long tail, lásd [48])[40]

Hátrányai

- **Megkülönböztethetetlenség problémája:** Ha két termék jellemzői azonosak (a két n -es megegyezik), a rendszer számára megkülönböztethetetlenek (Pl. egy jól és egy rosszul megírt tudományos cikk ugyanabban a témaiban).[40]
- **Kicsi a diverzitás:** Hasonló elemeket ajánl (lásd a 3.6.2. fejezet).

Tartalommalapú ajánló rendszer formális leírása

Egy adott felhasználó számára egy termék hasznossága ♦ A hasonló termékek ugyanazon felhasználó által megadott hasznosságaiból becsüljük és a legnagyobb becslés értékűt ajánljuk.

$$\hat{u}(c, s) \leftarrow \forall_u(c, s_i) \text{alapján, ahol: } s \sim s_i$$

[40]

Az ajánlás menete ♦

1. Az egyes elemekhez *tartalmi profilt* készítünk:

$$\forall_s \in S: s \rightarrow Content(s)$$

2. Az értékelt elemek alapján *felhasználói profilt* hozunk létre:

$$\forall_c \in C \{(Content(s), u(c, s)) \mid s \in S_c\} \rightarrow ContentBasedProfile(c)$$

3. Ezek alapján *becsüljük az ismeretlen hasznosságokat*:

$$\hat{u}(c, s) = score(Content(s), ContentBasedProfile(c))$$

[40]

Dokumentum profil készítése ♦ Például lehetséges heurisztikára egy publikációban:

- Jelöljük a rendszerben nyilvántartott *kulcsszavakat*:

$$\sigma_1, \sigma_2, \dots, \sigma_k$$

- Az s_i dokumentumban σ_j *kulcsszó gyakorisága*:

$$f_{i,j}$$

- Egy *dokumentum profilja* legyen az a vektor, mely az összes kulcsszó $w_{i,j}$ fontosságait, súlyát tartalmazza erre a dokumentumra:

$$Content(s_i) = \vec{w}_{s_i} = (w_{i,1}, w_{i,2}, \dots, w_{i,k})$$

- Egy *kulcsszó súlyának* meghatározása többféle módon definiálható, de például arányos lehet a normalizált előfordulási gyakoriságával:

$$TF_{i,j} = \frac{f_{i,j}}{\max_{z} f_{i,z}}$$

- Nem túl informatívak azok a kulcsszavak, melyek túl sok dokumentumban fordulnak elő, ezek súlyát hivatott csökkenteni az *inverse-document-frequency* tényező:

$$IDF_j = \log \frac{n}{n_j}, \text{ ahol } n_j = |\{i \mid f_{i,j} > 0\}|$$

- Így a *fontosság*:

$$w_{i,j} = TF_{i,j} \cdot IDF_j$$

[40]

Felhasználó profil készítése ♦ A felhasználó profilja lehet egy vektor, amely a múltban elolvasott vagy megvásárolt dokumentumok alapján az adott felhasználó *egyes kulcsszavakra vonatkozó* $v_{i,j}$ *preferenciáit* tartalmazza:

$$ContentBasedProfile(c_i) = \vec{v}_{c_i} = (v_{i,1}, v_{i,2}, \dots, v_{i,k})$$

A preferenciák meghatározása történhet a már értékelt dokumentumok w_{s_i} vektora-inak valamelyen átlagolásával, így a v preferenciavektor szintén tekinthető egy *TF-IDF* (Term Frequency-Inverse Document Frequency) vektornak.[40]

Hasznosságbecslés ♦ A hasznosságot például definiálhatjuk a v és w vektorok által bezárt szög koszinuszaként (Koszinusz hasonlósági függvény látható a 23. oldalon):

$$\begin{aligned}\hat{u}(c_i, s_l) &= \text{score}(\text{Content}(s_l), \text{ContentBasedProfile}(c_i)) = \\ &= \cos(v_{c_i}, w_{s_l}) = \frac{v_{c_i} \cdot w_{s_l}}{\|v_{c_i}\| \|w_{s_l}\|} = \\ &= \frac{\sum_{j=1}^k v_{i,j} \cdot w_{l,j}}{\sqrt{\sum_{j=1}^k v_{i,j}^2} \cdot \sqrt{\sum_{j=1}^k w_{i,j}^2}}\end{aligned}$$

3.4.3. Dimenzió csökkentés alapú ajánlás, mátrix faktorizáció

A mátrix faktorizáció (matrix factorization) a kollaboratív szűrés egy megközelítési módja. Ahogy a 3.4.1. fejezetben is olvasható, a *felhasználó-elem interakció* mátrixot két kisebb dimenziójú mátrix szorzatára bontják, lásd a 3.3. ábra. Ezek a módszerek a „Netflix-díj” (Netflix prize)⁴ végett lettek széleskörben ismertek a hatásfokuk miatt. Ennek nyertes pályázata számos *SVD*-modellt tartalmazott, beleértve a *Restricted Boltzmann gépekkel kevert SVD++t*[42, 55]

A 3.4.1. fejezetben azt is láthatjuk, hogy ez a kollaboratív ajánlók egy *modell alapú megközelítése*, amely egyaránt jól orvosolja a *skálázhatósági problémát* és a *zajos* és *ritka adatkészlet* problémáját is.

Mátrix Faktorizáció

A mátrix faktorizáció, más néven *mátrix dekompozíció* (matrix factorization, matrix decomposition), egy hatékony módszer a faktorterekben lévő felhasználók és elemek ábrázolására. A felhasználó-elem interakciók az f dimenziós látens faktortérben vett skaláris szorzatként vannak modellezve. minden elemhez egy $q_i \in \mathbb{R}^f$ vektor, míg minden felhasználóhoz egy $p_u \in \mathbb{R}^f$ vektor kapcsolódik. A p_u vektor elemei a felhasználók elemek iránti preferenciáit tükrözik, függetlenül attól, hogy ezek pozitív vagy negatív érdeklődést jelentenek. Így a két vektor skaláris szorzata közelíti meg a felhasználók valamely konkrét i elemre vonatkozó értékelését, amit $r_{u,i}$ -ként jelölünk a becslés során.[55, 56]

$$r_{u,i} \approx q_i \cdot p_u$$

⁴Egy nyílt verseny volt a filmek felhasználói értékelésének előrejelzésére szolgáló legjobb kollaboratív szűrési algoritmus megtalálására.

[55]

A felhasználó-elem interakciókat leíró vektorok leképzése után viszonylag egyszerű feladat a fenti egyenlet segítségével megbecsülni a felhasználó által az elemre adott értékelést.[55]

Szinguláris érték szerinti felbontás (SVD)

A *szinguláris érték szerinti felbontás* (singular value decomposition vagy SVD) széleskörben használt ajánlórendszerben, azon belül a kollaboratív szűrésben. Segít meg találni a *látens változókat* – a statisztikában a látens, vagy más néven háttér változók (angolul latent factors), olyan változók, amelyekre csak közvetve lehet következtetni egy matematikai modell segítségével, más, közvetlenül megfigyelhető vagy mérhető változók alapján.[57] Mivel ez az eljárás igen összetett, a formális leírására nem térek ki, kiváló referenciák találhatók, például amiket én is használók ennek az 3.4.3. alfejezetnek a megírásához [55, 56, 58].

Ajánlás menete, film értékelések becslésére

Értékelés mátrix létrehozása ♦ Felhasználó-elem R mátrix létrehozása, ahol:

- a sorok a felhasználókat,
- az oszlopok a filmeket
- és a mátrix elemei a felhasználók által, a filmekre adott értékeléseket

reprezentálják. Fontos, hogy ez a mátrix tele van üres, vagy nem informatív értékekkel (sparse data), amiknek a megbecslése a célunk, máskülönben értelmetlen lenne az folyamat, a végcél ugyanis a legnagyobb értékre becsült elem vagy elemek meghatározása.

SVD alkalmazása ♦ A szinguláris érték szerinti felbontás az eredeti R mátrixot 3 hasonló mátrixra bontja:

- U felhasználó mátrix: Ez a mátrix a felhasználók és a háttér változók kapcsolatát reprezentálja. A sorok a felhasználókat, míg az oszlopok a háttér változókat jelölik.
- Σ szinguláris érték mátrix: Diagonális mátrix⁵, melynek elemei mutatják a háttér változók fontosságát, súlyát, ezek a *szinguláris értékek*. Az elemek csökkenő sorrendben vannak rendezve, jelezve, hogy az első pár faktor a legfontosabb.
- M^T film mátrix transzponáltja: Ez a mátrix a filmek és a háttér változókat kapcsolatát mutatja. Az oszlopok a filmeket, míg a sorok a háttér változókat jelzi.

⁵Diagonális vagy diagonálmátrix: olyan $A = (a_{i,j})$ négyzetes mátrix, melynek minden főátlón kívüli eleme nulla[59]

Dimenzió csökkentés ♦ Az SVD egyik legnagyobb előnye a *dimenzió csökkentés*, mivel csak a legnagyobb k szinguláris értékeket tartja meg a Σ mátrixból és az ezekhez tartozó vektorokból az U és M^T mátrixokban. Ez a lépés kulcsfontosságú a dimenziócsökkentésben és az adatok közelítésében. Ennek gyakorlati haszna:

- **Hatókonyiság növelése**
- **Zajcsökkentés**: a kisebb szinguláris értékek gyakran a zajnak vagy kevésbé fontos információknak felelnek meg.
- **Értelmezhetőség**

Hasznosságbecslés ♦ Ahhoz, hogy megbecsüljük, a felhasználó számára vajon mennyire hasznos egy termék, jelen esetben egy film, vegyük

- a felhasználó vektorát az U mátrixból
- a szinguláris érték mátrixot Σ
- és a film vektorát a M^T mátrixból.

$$R = U\Sigma M^T$$

Ez az eredmény az eredeti értékelési mátrix közelítését eredményezi, a hiányzó értékekre kitöltött becslésekkel.

Egy u felhasználó egy konkrét m film értékelésének becslését megkaphatjuk:

$$R_{u,m} = U_u \cdot M_m^T$$

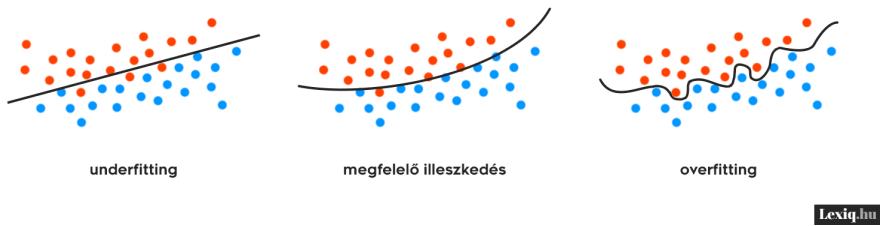
3.5. Az adat előkészítése a modell tanítására és tesztelésre

A tisztítás után az adatot elő kell készíteni a modell tanítására és tesztelésére. Ehhez két vagy három diszjunkt részre bontjuk a rendelkezésre álló adatokat (lásd 2.3.3. fejezet).

3.5.1. Túllilleszkedés

„A gépi tanulásban a *túltanulás* vagy *túllilleszkedés* (overfitting) azt jelenti, hogy a rendszer által kialakított modell túlságosan alaposan követi a tanításra használt adathalmazt, azaz nem csak a lényegi információt, hanem az adott véges adathalmaz egyedi furcsaságait is megtanulja.

Túltanulás esetén a modell jól működik a tanításra használt adathalmazon, de rosszabbul teljesít, ha más, korábban nem látott adathalmazra alkalmazzuk.”[60]



3.4. ábra. Overfitting ábrázolása egy adathalmazon[60]

3.5.2. Adatfelosztásos validáció (split-validation)

A tanító készlet általában az összes adat 70-80 míg a tesztelő 20-30 százalékát jelenti. Míg a tanító készletet arra használjuk, hogy betanítsuk a modellt, a tesztelő készletet arra, hogy a modell pontosságát mérni tudjuk.

3.5.3. K-szoros keresztképzés (k-fold cross validation)

Itt az adatkészletet csakugyan elosztjuk k egyenlő részre (fold), amiből 1 lesz a tesztkészlet⁶. Így tehát a $k-1$ darab tanító készleten betanítjuk, majd az így kapott modellt, ezen a maradék 1 készleten leteszteljük. Ezt az eljárást addig ismételjük, amíg az összes készletet fel nem használjuk tanító és tesztkészletként egyaránt. Az így kapott eredményeket ezután összesítjük és ennek átlagát vesszük, így egy átfogó értékelést alkotunk a modell teljesítményéről. Ez az átlagolt érték adja meg, hogy a modell általánosítási képessége várhatóan milyen lesz ismeretlen adatokon. Ezen kívül, a különböző iterációk során kapott eredmények szórása is fontos információkat szolgáltathat a modell stabilitásáról és megbízhatóságáról.⁷ Ez több számítási kapacitást igényel, viszont elkerüljük a túlilleszkedést (lásd 3.5.1. fejezet).

3.6. Algoritmusok kiértékelése, összehasonlítása és metrikák

3.6.1. Pontossági mérőszámok

Számos algoritmus áll rendelkezésre, hogy megkapjunk egy mérőszámot mely az ajánló rendszerünk pontosságát tükrözi, ebből bemutatok néhányat. Ebben az alszekcióban

⁶ A k részből egynél több is szolgálhat tesztkészletként

⁷ Amikor a tesztkészlet minden elemét egyetlen elemből áll, akkor ezt *Leave-One-Out Cross-Validation* (LOOCV) néven ismerjük, míg a *Leave-P-Out Cross-Validation* (LPOCV) esetén a tesztkészlet P elemű, ahol $P > 1$.

lévő egyenletekhez felhasznált irodalom: [61, 62]

A matematikai formulákban lévő jelölések:

- n jelöli a tesztkészletben lévő *elemek számát*,
- x a *megfigyelt értékek*, azaz a felhasználók által adott értékelések, x_i az i -edik megfigyelt érték,
- y a rendszerünk által készített *becslések*, y_i az i -edik becslés.

Hibák mérése

Általánosságban elmondható az alábbi metrikáakra, hogy minél kisebb a kapott érték, annál pontosabb becsléseket készít a rendszerünk.

Átlagos hiba (ME) ♦ Angolul *Mean Error (ME)* vagy *Bias (B)*.

Vegyük a tesztkészletben lévő, a rendszerünk által készített becslések és a felhasználó által adott értékelések különbségét és ennek számítsuk ki áz átlagát.

$$\text{ME} = \mathbf{B} = \frac{\sum_{i=1}^n (y_i - x_i)}{n}$$

Ez a legegyszerűbb mérőszám. Nem méri a hiba abszolút értékét, tehát nullát úgy is el lehet érni, hogy pozitív és negatív irányba is eltérhet a becslésünk és a megfigyelés különbsége. Más szavakkal a rendszerünk túl- és alulbecsülheti az egyes elemeket.

Átlagos abszolút hiba (MAE) ♦ Angolul *Mean Absolute Error (MAE)*.

Vegyük a tesztkészletben lévő, a rendszerünk által készített becslések és a felhasználó által adott értékelések különbségének abszolút értékét, és ennek számítsuk ki áz átlagát.

$$\text{MAE} = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$$

Az előző (ME) hibáját igyekszik korrigálni, tehát a különbségek abszolút értékét veszi, ezáltal minden eltérést hibának vesz, legyen az pozitív vagy negatív irányban, de ez az egyik hibája is egyben, nem mutatja, hogy alul vagy túlbecsültük az elemeket.

Átlagos négyzetes hiba (MSE) ♦ Angolul *Mean Squared Error (MSE)*.

Vegyük a tesztkészletben lévő, a rendszerünk által készített becslések és a felhasználó által adott értékelések különbségének négyzetét, és ennek számítsuk ki áz átlagát.

$$\text{MSE} = \frac{\sum_{i=1}^n (y_i - x_i)^2}{n}$$

Ez sem veszi figyelmen kívül hagyja, hogy alul- vagy túlbecsültük az elemeket. A négyzetre emelés miatt érzékenyebb a nagyobb eltérésekre.

Átlagos négyzetes hiba gyöke (RMSE) ♦ Angolul *Root Mean Squared Error (RMSE)*.

Vegyük a tesztkészletben lévő, a rendszerünk által készített becslések és a felhasználó által adott értékelések különbségének négyzetét, és ennek számítsuk ki áz átlagát, majd ennek vegyük a négyzetgyökét.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^n (y_i - x_i)^2}{n}}$$

Ez az előző kifejezés (MSE) négyzetgyökeként értelmezhető. Az eredeti mennyiséggel (azaz a megfigyelt változóval) azonos mértékegységgel rendelkezik. Például ha:

$$x = [5,5,4,4,3,2,5] \text{ és } y = [2,5,3,2,1,5,1]$$

akkor

$$\text{MSE} = 6.1429$$

$$\text{RMSE} = 2.4785$$

RMSE jobban tükrözi a tényleges átlagos hiba nagyságát az eredeti mértékegységen.

Top-N Találati arány

A *találati arány* (hit-rate) eredetileg az üzleti teljesítmény mérőszáma, mely hagyományosan az értékesítésekhez kapcsolódik.[63] Ezt a metrikát gyakran használják ajánlórendszerek teljesítményének a mérésére. A felső N találati arány (top-N hit rate) azt mutatja, hogy a rendszer által ajánlott tételek közül hány esetben szerepel legalább egy, felhasználó által előnyben részesített tételek között.[65]

Számításának menete ♦

1. **Adatok előkészítése:** Osszuk fel az adatkészletünket tanító és tesztelő készletekre. Az adatkészletből távolítsunk el *egy darab*, a felhasználó által preferált elemet, ez lesz a tesztkészletünk⁸.
2. **Modell tanítása:** tanítsuk be az ajánlórendszeret a tanítókészletet felhasználva.
3. **Top-N ajánlások generálása:** A tesztkészlet minden felhasználójához generálunk egy top-N ajánlási listát.

⁸Leave-One-Out cross validation vagy LOOCV

4. Találatok számítása : Vizsgálunk meg minden felhasználót, hogy a tesztkészlet eleme megjelenik-e az ajánlott N termékben, ha igen, az egy *találat* (hit).

[64]

Ha például van 100 felhasználónk és 90 felhasználónál szerepelt a megfigyelt elem a *legjobb 10* ajánlásban, akkor a Top-10 Hit Rate 90%-os.

3.6.2. Viselkedési metrikák

Léteznek úgynevezett viselkedési metrikák, melyek nem a pontosságot, hanem egyéb fontos tulajdonságait mutatja az ajánlórendszerünknek. Ebben a részben felhasználtam a [66] található információkat.

Ilyen metrikák például:

- **Lefedettség** (coverage): Egy százalékos arány, mely azt mutatja meg hogy a tanító készletből mennyi elemet tud ajánlani a tesztkészleten.[68]
- **Diverzitás** (diversity): Azt méri, hogy mennyire változatosak az ajánlott elemek. Egy magas diverzitású ajánlórendszer széles választékot kínál, ami segíthet felfedezni új érdeklődési területeket, így növelte a felhasználói elégedettséget. A diverzitás növelése azonban csökkentheti az ajánlások relevanciáját, ezért fontos megtalálni az egyensúlyt.
- **Újdonság** (novelty): Magas újdonsággal azok az új, vagy hosszú farokban⁹ lévő termékek bírnak, amelyekkel kevés felhasználó lépett interakcióba, a népszerű termékeket viszont alacsony újdonsággal lehet jellemzni.[72]
- **Véletlenszerű felfedezés** (serendipity): Ez a metrika arra utal, amikor a rendszer olyan elemeket ajánl a felhasználónak, amelyek nem csak relevánsak, hanem meglepőek is, azaz olyan tartalmakat, amelyeket a felhasználó valószínűleg nem talált volna meg magától, de mégis értékel. A számszerűsítésének legyegyszerűbb módja a top-N ajánlott cikk átlagos népszerűsége.[67]
- **Lemorzsolódás** (churn) és **Reakciótérfogat** (responsiveness): A *lemorzsolódás* azt méri, hogy új értékelések után, milyen gyakorisággal változnak az ajánlások. A *reakciótérfogat* az ilyen változások sebességét mutatja. [69]

Ezeknél a mérőszámoknál nem lehet feketén-fehéren kijelenteni, hogy a nagyobb érték az jobb rendszert eredményez. Ezek úgymond az ajánlórendszerünk stílusát, szófisztiáltságát jellemzik. Fontos lehet, hogy az ajánlórendszerünk tudjon sokszínű ajánlásokat tenni, ugyanakkor ennek a mértékének a meghatározása főleg valós felhasználókon történő teszteléssel lehet meghatározni (lásd 3.6.3. fejezet).

⁹ A hosszú farokról bővebben [48]

3.6.3. Online A/B Teszt

Az *A/B tesztelés* más néven kettős tesztelés (split testing vagy bucket testing) egy olyan módszer, melyet a *felhasználók viselkedésének* kutatására használnak. Általában egy adott dolog (például termék, weboldal, ajánlórendszer, stb.) kettő (A/B) vagy több változat összehasonlítására szolgál, általában azért, hogy kiválasszuk, melyik verzió teljesít jobban a *valós felhasználók viselkedése alapján*.[70]

Lépések:

1. **Adatgyűjtés:** Mielőtt megnéznék, hogy a rendszerünk más verziója hogyan teljesít, érdemes lehet a jelenlegi állapotáról információkat gyűjteni.
2. **Hipotézis felállítása:** Egy hipotézis megfogalmazása, hogy a változatok között melyik és miért lehet jobb, miért van szükségünk a fejlesztésre, módosításra.
3. **Változatok létrehozása:** Készítsünk kettő vagy több változatot, melyek csak egy-egy dologban térnek el egymástól.
4. **Kísérleti csoportok kiválasztása:** A változatokat véletlenszerűen kiválasztott csoportoknak mutatjuk be.
5. **Adatgyűjtés az összes változatról:** Figyeljük meg a felhasználók viselkedésének változásait a különböző változatoknál.
6. **Elemzés:** Megállapítjuk, hogy melyik változat teljesített jobban a hipotézisben megfogalmazott kiválasztott nézőpontok alapján.
7. **Döntéshozatal:** Eldöntjük melyik verziót szeretnénk bevezetni.

[71]

Mivel *valós felhasználókon* teszteljük a termékünk különböző verzióit, teljesen objektívan tudunk döntést hozni a változtatást illetően. Ugyanakkor hogy ezek az eredmények megbízhatóak legyenek elengedhetetlen, hogy megfelelően válasszuk ki a kísérleti csoportjainkat valamint ha valóban a verziók csupán csak egy-egy dologban térnek el, akkor nehezebbé válik a komplexebb változtatások tesztelése.

4. fejezet

Specifikáció

4.1. Feladat specifikáció

Egy olyan alkalmazást tűztem ki a szakdolgozatom céljául, amivel szeretném demonstrálni egy modern webalkalmazás felépítését, népszerű eszközök használatával, valamint különböző szolgáltatások együttműködését, hogy hogyan is kerülnek a felhasználók elé az ajánlórendszer által készített ajánlások, egy egyszerű *filmajánló* példáján keresztül. A hangsúlyt szerettem volna az alkalmazás felépítésére helyezni, ezért ezt csak egy *MVP*-nek¹ nevezném, mintsem egy kész alkalmazásnak. Ezért hiányoznak belőle olyan funkcionálisok, mint például az admin jogosultság, felhasználó vagy film adatainak módosítása, filmek értékelése.

4.2. Rendszerterv

A *rendszerterv* (system design) az a folyamat, ahol definiáljuk, mely feltételeknek kell megfelelnie az elkészült alkalmazásunknak és azt, hogyan tervezzük megvalósítani azt.[73, 74]

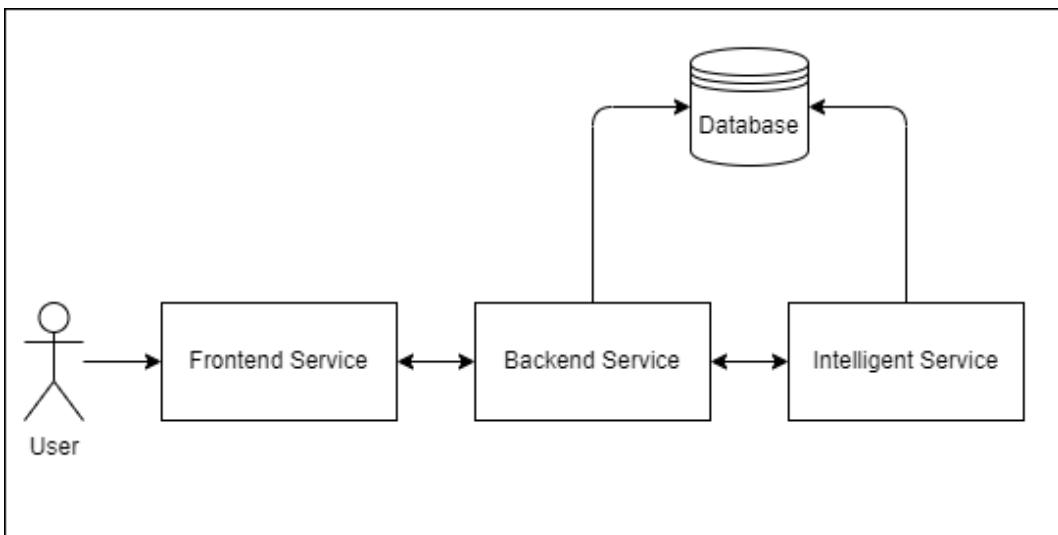
Ez egy kritikus lépés, mert az online szolgáltatásoknál széles skálán mozoghatnak a felhasználók, akiket ki kell szolgálnunk. Előfordulhat, hogy weboldalunkat csak néhányan látogatják, de például, a *Facebook napi felhasználóinak (daily active users)* száma 2.064 milliárd.² A rendszertervünknek tehát döntő szerepe lesz abban, hogy milyen felhasználói élményt nyújtunk, mennyire állunk azok rendelkezésére, így tehát, hogy sikeres lesz a projektünk, vagy kudarcba fullad.

¹ Minimum Viable Product (MVP): „Minimálisan Életképes Terméket jelent. A legkisebb olyan funkcionálitással rendelkező termék vagy szolgáltatás, amelyre visszajelzést tudunk kapni a potenciális felhasználóktól, ugyanakkor még nem feltétlenül piacképes.”[76]

² Forrás: <https://www.demandsage.com/facebook-statistics/>

4.2.1. Architektúra

Az alkalmazásomhoz *szolgáltatásorientált architektúrát* választottam. A *szolgáltatásorientált architektúra* (angolul Service-Oriented Architecture, röviden SOA) egy programozási módszer, aminek lényege, hogy az egyes szolgáltatások egy hálózatban helyezkednek el, és egy protokoll (jelen esetben *HTTP*) által meghatározott módon kommunikálnak. A szolgáltatások különálló egységek, amik távolról is, valamint egymástól függetlenül is elérhetők, használhatók, frissíthetők, újra kombinálhatók.[75] Ez nagyon hasznos, hiszen ha más kliensoldali technológiát szeretnénk használni (például egy másik népszerű választás az Angular vagy Vue.js), akkor nem kell drasztikus változtatásokat végeznünk a frontenden, elég, ha megírjuk az új szolgáltatást, és ahhoz hozzá kapcsoljuk a már meglévő backendet, de ugyanez elmondható az összes komponense re. Valamint ezek a komponensek külön-külön is skálázhatók, ha szükséges, például *Kubernetes*³ használatával.



4.1. ábra. Magas szintű rendszerterv

A 4.1. ábrán igyekeztem egyszerűen szemléltetni a rendszerem tervét. A felhasználó (User) a *Frontend Service*-szel (továbbiakban frontend vagy front-end szolgáltatás) van kizárolagos kapcsolatban. A frontend *HTTP*⁴ protokollt használva kommunikál a *Backend Service*-szel (továbbiakban backend vagy back-end szolgáltatás), olyan kéréseknél mint például a bejelentkezés, regisztráció, ajánlások kérése. A backend kapcsolatban van az *Intelligent Service*-szel (továbbiakban intelligens szolgáltatás), mely szolgáltatás feladata, hogy előállítsa az ajánlásokat a felhasználó részére, ez a kommunikáció is *HTTP*⁴ protokollon keresztül történnek. Mind a *backend*, mind az *intelligens szolgáltatás* kapcsolatban van az adatbázissal (Database), ahol a felhasználókról, filmekről, ajánlásokról tárolunk információkat (lásd 4.2.2. fejezet).

³<https://kubernetes.io/>

⁴ Éles környezetben biztonsági okokból érdemes *HTTPS*-t használni.

4.2.2. Adatbázis terv

Az adatbázis megtervezésénél meg kell határozni, hogy milyen formában fogjuk tárolni az adatot, ez halmozottan fontos, ha strukturált adatbázist használunk, például PostgreSQL-t. Az 1.6.1. fejezetben olvasható, miért a *PostgreSQL* mellett döntöttem.

Egy meglévő, a *GroupLens Research*⁵ által összegyűjtött adatkészletet, a *MovieLens*⁶-t (azon belül is az *ML latest small* adatkészletet) használom, ami eredetileg CSV (Comma-Separated Values) formátumban van tárolva, ezt PostgreSQL adatbázisba migrálom, tehát *strukturált tabuláris adatot* alkalmazok. Ehhez a migrációhoz írtam pár Python szkriptet⁷.

- *separate-release-year-from-title.py*: Ezzel elkülönítem a *title* részből a *release-Year*-t.
- *generate-sql-script-to-create-dummy-users.py*: Létrehozunk egy *sql* szkript fájlt, mellyel az adatbázist feltöltöm tesztelő felhasználókkal. Ezek nem valós felhasználói az alkalmazásnak, az adatkészletben a *ratings.csv* fájlban az értékelésekhez van egy *userId* társítva. Ez szükséges ahhoz, hogy tesztelhessük az alkalmazást.
- *generate-sql-script-to-insert-data-into-movies-genres-and-movie-genres.py*: Ezt használva létrehozok egy *sql* szkriptet, mely feltölti az adatbázist a *movies.csv* fájlban található filmeket, műfajokkal.
- *generate-sql-script-to-insert-data-into-ratings.py*: Végül a filmeket és felhasználókat felhasználva létrehozunk egy *sql* szkriptet, ami az értékelésekkel egészíti ki az adatbázisunkat.

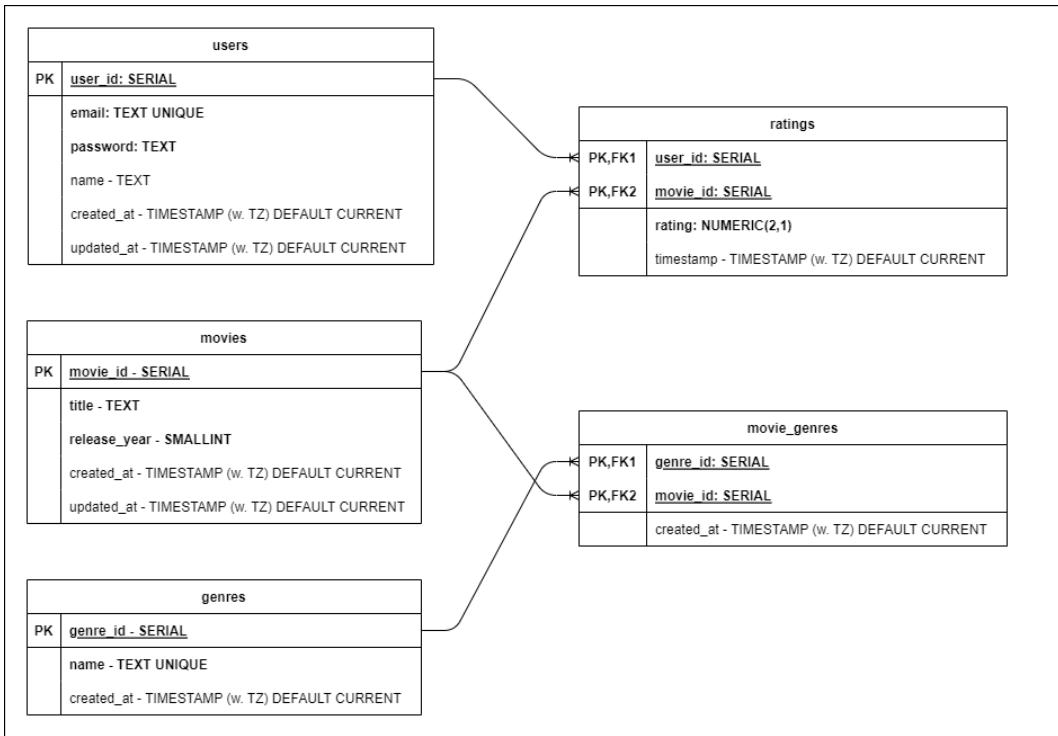
Ezekre a szkriptekre azért volt szükség, hogy *harmadik normálformában* (3NF) tárolhassam az adatokat az adatbázisunkban (CSV fájlok helyett). A 3NF által csökkenjük az adatok redundanciáját, megakadályozzuk azok inkonziszenciáját, javítjuk az adatintegritást, és optimalizáljuk a lekérdezések teljesítményét. Az így kapott adatbázis tervünk ER Diagramja⁸ látható a 4.2. ábrán.

⁵ GroupLens Research hivatalos oldala: <https://grouplens.org>

⁶ MovieLens hivatalos oldala: <https://movielens.org>

⁷ A Python szkriptek megtalálhatók az *applikáció/intelligent-service/data/scripts* mappában

⁸ ER-modell: <https://hu.wikipedia.org/wiki/ER-modell>



4.2. ábra. ER Diagram

Felhasználók

A felhasználóról csak a legfontosabb adatokat tároljuk, melyek elegendők azok megkülönböztetésére, azonosítására (autentikáció, lásd 4.2.4. fejezet).

A regisztrációhoz, belépéshez kötelező megadni egy email címet (email) és egy jelszót (password) (amit majd hashelünk⁹ és titkosítunk), valamint opcionálisan megadható egy felhasználónév (name). Nyomon szeretnénk követni, hogy mikor hozták létre (created_at), valamint mikor módosították (updated_at) utoljára a felhasználót. minden felhasználóhoz szeretnénk hozzá társítani egy automatikusan létrehozott ID-t (user_id), amit elsődleges kulcsként fogunk használni (Primary Key).

Mező	Típus	Egyéb megkötés
user_id	SERIAL	Primary Key
email	TEXT	Unique, Not Null
password	TEXT	Not Null
name	TEXT	-
created_at	TIMESTAMP TZ ¹⁰	DEFAULT_CURRENT
updated_at	TIMESTAMP TZ ¹⁰	DEFAULT_CURRENT

4.1. táblázat. Felhasználók tábla

⁹ „A hashelés azt a folyamatot jelenti, amelynek során egy változó méretű bemenetből egy fix méretű kimenetet állítunk elő.” Forrás: <https://academy.binance.com/hu/articles/what-is-hashing>

¹⁰ TZ-t a ”WITH TIMEZONE”, rövidítésére használom helytakarékkossági okokból

Filmek

A filmek (movies) tábla is minden össze a film címéből és a kiadásának évből tevődik össze, ehhez is hozzárendeltem egy automatikusan létrehozott ID-t (movies_id), amit elsődleges kulcsként fogunk használni valamint a létrehozásának és módosításának dátumát megőrző (created_at és updated_at) mezőket.

Mező	Típus	Egyéb megkötés
movie_id	SERIAL	Primary Key
title	TEXT	Not Null
release_year	SMALLINT	Not Null
created_at	TIMESTAMP TZ ¹⁰	DEFAULT_CURRENT
updated_at	TIMESTAMP TZ ¹⁰	DEFAULT_CURRENT

4.2. táblázat. Filmek tábla

Műfajok

A műfajok (genres) táblában egy automatikusan létrehozott ID-t (genre_id) és a létrehozásának dátumát (created_at), valamint a műfaj megnevezését (name) tároljuk.

Mező	Típus	Egyéb megkötés
genre_id	SERIAL	Primary Key
name	TEXT	Not Null, Unique
created_at	TIMESTAMP TZ ¹⁰	DEFAULT_CURRENT

4.3. táblázat. Műfajok tábla

4.2.3. Film Műfajok

A filmek és műfajok közötti kapcsolatot a movie_genres táblában rögzítjük több-több (many-to-many) kapcsolattal, mivel egy filmhez több műfaj is tartozhat, és egy műfajhoz több film is kapcsolódhat. Ez a tábla egy úgynevezett *kapcsolótábla* (junction table), amely lehetővé teszi a filmek és műfajok közötti több-több kapcsolat reprezentálását. A genre_id és a movie_id is részét képzi az elsődleges kulcsnak (Primary Key) valamint egy-egy idegen kulcsok (Foreign Key) melyek a filmek és a műfajok táblákat kapcsolja hozzá ehhez a táblához.

Mező	Típus	Egyéb megkötés
genre_id	INT	Primary Key, Foreign Key
movie_id	INT	Primary Key, Foreign Key
created_at	TIMESTAMP TZ ¹⁰	DEFAULT_CURRENT

4.4. táblázat. Felhasználók tábla

Értékelések

Az értékelések (ratings) tábla is egy *kapcsolótábla* (junction table), mely *több-több* (many-to-many) kapcsolatban köti össze a filmeket és a felhasználókat, akik értékelték azt. Itt a két genre_id és a movie_id együtt teszik ki az elsődleges kulcsot, valamint egy-egy idegen kulcsok. Ezen kívül tároljuk magát az értékelést 2 számjegyű formátumban, melyből egy a tizedes jegy, azaz az értékek lehetnek 0.0-tól 9.9-ig (például 2.5 helyes, de 25 hibás). Tárolunk még egy timestamp mezőt, ami az eredeti adatkészletből származik.

Mező	Típus	Egyéb megkötés
genre_id	INT	Primary Key, Foreign Key
movie_id	INT	Primary Key, Foreign Key
rating	NUMERIC(2,1)	Not Null
timestamp	TIMESTAMP	DEFAULT_CURRENT

4.5. táblázat. Értékelések tábla

4.2.4. Authentikáció

Az *autentikáció* az a folyamat, amely során egy felhasználó vagy rendszer igazolja saját identitását egy másik rendszer felé. Ez történhet többféle képpen, például az általunk használt email cím-jelszó párossal, vagy például digitális tanúsítvánnyal, biometrikus azonosítással is.[77]

Az alkalmazásban első lépésként, a felhasználónak azonosítania kell magát az előírt email cím-jelszó párossal. Ha az azonosítás sikeresen megtörtént, a backend létrehoz egy *JWT tokent*¹¹, mely tartalmazza a felhasználónk ID-ját. Majd létrehoz egy *HttpOnly cookie-t*, amiben eltárolja ezt a tokent. A *HttpOnly cookie* azért biztonságosabb mint a „hagyományos” (nem *HttpOnly*) *cookie* vagy egyéb tárolási mód, mert korlátozza a cookie-hoz való hozzáférését a kliensoldali szkripteknek. Ez növeli *XSS*¹² támadások elleni védelmet. A *HttpOnly cookie-nál* a cookie tárolja a:

- nevét (name)
- értékét (value)
- domaint (domain)
- lejárat (expires / max age)
- útvonalt (path)

¹¹ JWT-ről bővebben: <https://jwt.io/>

¹² XSS támadásról bővebben: https://en.wikipedia.org/wiki/Cross-site_scripting

- lejárati dátumát (expires)
- méretét (size)
- egy boolean értékként, hogy HttpOnly cookie-e
- biztonságos-e (secure): ez annyit jelent, hogy HTTPS protokollt használva lett-e küldve, ami csökkenti a *közbeékelődéses támadás*¹³ (man-in-the-middle) kockázatát
- egy úgynevezett *SameSite* mezőt, melynek értéke lehet:
 - **Strict:** Az alkalmazásban ezt választottam, mivel ez a Cross-site request forgery (CSRF)¹⁴ ellen óvja a felhasználót. Ezzel lényegében azt mondjuk a böngészőnknek, hogy csak a cookie-t beállító webhelyről származó kérésekben szerepeljen ez a cookie.
 - **Lax**
 - **None**
 - **Nem definiált:** Ebben az esetben *Chrome 80-as verzójától* Lax értékként értelmezi a böngésző (olyan böngészőknél, melyek ezt a motort használják).

Ha ki szeretnénk jelentkezni, a *backend* beállítja 0-ra a HttpOnly cookie *maxAge* attribútumát, tehát a cookie azonnal le fog járni, így használhatatlan lesz a továbbiakban.

Ugyan az alkalmazásban csak ennyi autentikációt implementálok, érdemes megemlíteni pár másik nézőpontot. Az általam írt *docker-compose.yml* fájl fejlesztéshez és teszteléshez készült, ezért a dockeren kívül is elérhetőek a *backend* és az *intelligens szolgáltatás* API végpontjai. Éles környezetben azonban eldönthetjük, hogy az intelligens szolgáltatás végpontjait publikáljuk-e. Ha nem, akkor nincs is más dolgunk, mert a docker egy zárt környezet, de az ezen belül lévő konténerek tudnak egymással kommunikálni. Ellenkező esetben megfontolandó, hogy ezeket a végpontokon implementálunk autentikációt, autorizációt¹⁵.

¹³ Közbeékelődéses támadásról bővebben: [78]

¹⁴ CSRF-ről bővebben: <https://www.educative.io/answers/what-is-cross-site-request-forgery-csfr>

¹⁵ Míg az autentikáció azt nézi, ki végzi a műveleteket, az autorizációnál azt nézzük, hogy ennek az entitásnak *van-e jogosultsága* az adott műveletek elvégzéséhez.

5. fejezet

Fejlesztés bemutatása

Az ebben a fejezetben bemutatottak csak részletek vagy reprezentációk. Az alkalmazás megtalálható a <https://github.com/fepu08/szakdolgozat> GitHub tárolóban az **mvp** mappában. minden esetben, ha hivatkozok egy forráskód fájlra (például *./intelligent-service/app.py*) akkor az az *mpv* mappától számított relatív útvonal.

5.1. Intelligens szolgáltatás Benchmark modulja

A következő 5.2. és 5.3 fejezetekben az eltelt idő méréséhez, az intelligens szolgáltatásban implementáltam egy *Benchmark*¹ modult, mely az alkalmazáson belül az *./intelligent-service/utils/Benchmark.py* fájlban található. Az itt található *TimeBenchmark* osztály használatával – mely forráskódja az 5.1. kódban látható – mérem az eltelt időt.

```
1 class TimeBenchmark:
2     def __init__(self):
3         self.start_time = None
4         self.end_time = None
5         self.elapsed_time = None
6
7     def start(self):
8         if(self.start_time):
9             self.__clear()
10            self.start_time = time.time()
11
12     def stop(self):
13         self.end_time = time.time()
14         self.elapsed_time = self.end_time - self.
15             start_time
```

¹ Benchmark: számítógépes program, programkészlet vagy egyéb műveletek futtatása egy objektum relatív teljesítményének felmérése érdekében, általában számos szabványos teszt és próba futtatásával.[80]

```

15
16     def print_elapsed_time(self):
17         print(f"Elapsed time (seconds): {self.
18             elapsed_time}")
19
20     def __clear(self):
21         self.start_time = None
22         self.end_time = None
23         self.elapsed_time = None

```

5.1. kód. TimeBenchmark osztály

Ennek használata nagyon egyszerű, az 5.2. kódban látható.

```

1 timebenchmark = TimeBenchmark()
2 timebenchmark.start()
3
4 # ...
5
6 timebenchmark.stop()
7
8 timebenchmark.elapsed_time
9 timebenchmark.print_elapsed_time()

```

5.2. kód. TimeBenchmark használata

Először is példányosítani kell a *TimeBenchmark* osztályt (5.2. kód 1. sora), ami inicializálja a példány mezőit *None* értékre (5.1. kód 2-5. sorai). Azért érdemes itt példányokat használni a statikus metódusokkal ellentétben, mert ha egyszerre több helyen szeretnénk mérni az időt megtehetjük, egymástól függetlenül, különböző példányokat használva. Így, mikor az 5.2. kód 2. sorában meghívjük a példány *start* metódusát, az beállítja annak *start_time* mezőjének értékének az aktuális időt, Unix időbényeg formájában.² Ha be szeretnénk fejezni a mérést, akkor a példány *stop* metódusát kell hívnunk (5.2. kód 6. sora). Ahogy az 5.1. kód 13. sorában olvasható, ez a példány *end_time* értékét fogja megadni Unix időbényeg formájában ugyan úgy, mint a *start* metódusnál. Majd a 14. sorban az *end_time* és a *start_time* értékeket kivonva, megadjuk a *time_elapsed* mező értékét, mely a *start* és a *stop* között eltelt időt mutatja. Ezt kiírathatjuk a az 5.1. kód 14. sorában levő *print_elapsed_time* metódussal.

Ha újra szeretnénk indítani a mérést, megtehetjük a példány *start* metódusának a meghívásával. Ilyenkor az 5.1. kód 8. sorában ellenőrizzük, hogy volt-e korábbi futtatás, a példány *start_time* mezőjének vizsgálatával, ha ez igaz értéket ad vissza, akkor meghívjuk az osztály privát *__clear* metódusát, mely az 5.1. kód 19-22. soraiban látható. Ez visszaállítja a példány mezőinek értékét a konstruktorban definiáltak szerint, azaz

² Unix időbényeg: 1970.01.1. 00:00:00 UTC óta eltelt másodpercek száma, lebegőpontos formában.

None-ra.

5.2. Hiperparaméter optimalizálás

A *hiperparaméterekről* és azok optimalizálásáról a 2.5 fejezetben már írtam, ennek implementálását szeretném bemutatni a *Surprise* könyvtár használatával.

A *Surprise* könyvtár lehetőséget biztosít *hiperparaméter optimalizálásra* a *GridSearchCV* és *RandomizedSearchCV* nevű osztályok alkalmazásával. A szakdolgozatban az előbbit használom. Ez az osztály rácskereséses módszert használva az algoritmus pontossági mérőszámait számítja ki a hiperparaméterek különböző kombinációira, *keresztfelügyeleti validációt* (lásd a 3.5.3. fejezetben) használva. Az implementáció megtalálható az `./intelligent-service/hyperparameter_optimization.py` elérési úton. Egy ahoz hasonló kód részlet látható az alábbi 5.3. kódban. Referenciaként a dokumentációt használtam.[10]

```
1 param_grid = {
2     'n_factors': [50, 100, 150],
3     'n_epochs': [5, 10, 20],
4     'lr_all': [0.002, 0.005, 0.007],
5     'reg_all': [0.02, 0.05, 0.1],
6     'lr_bu': [0.002, 0.005],
7     'lr_bi': [0.002, 0.005],
8     'reg_bu': [0.02, 0.05],
9     'reg_bi': [0.02, 0.05],
10    'reg_pu': [0.02, 0.05],
11    'reg_qi': [0.02, 0.05]
12 }
13 gs = GridSearchCV(SVD, param_grid, measures=["rmse",
14                                         "mae"], cv=5)
15 gs.fit(dataset)
16
17 gs.best_score['rmse']
gs.best_params['rmse']
```

5.3. kód. Hiperparaméterek optimalizálása *GridSearchCV* használatával

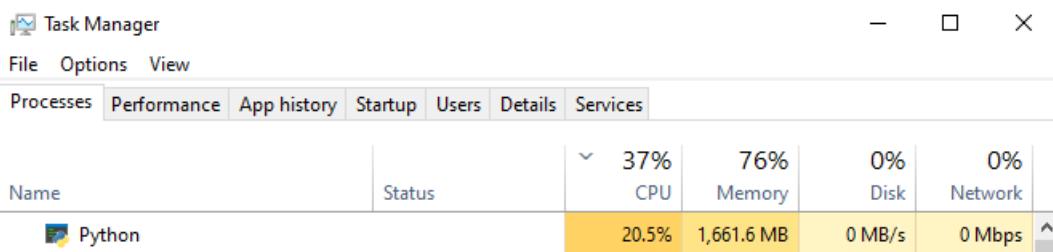
Az 5.3. kódban látható kód részlet egy *SVD* (lásd a 3.4.3. fejezetben) algoritmus számára keresi a legjobb paraméter párokat az 1-12. sorokban biztosított paraméterracs alapján. Ez az adatracs a modell különböző paramétereinek a lehetséges értékeit tartalmazza, mint például a faktorok száma (*n_factors*), vagy mint a tanulási ráta (*lr_all*). A 13. sorban példányosítjuk a *GridSearchCV* osztályt. Ennek megadjuk paraméterként

- az algoritmust (*SVD*),
- a paraméterracsot (*param_grid*),

- a mérőszámokat, melyeket használ az összehasonlításokhoz (`measures=["rmse", "mae"]`),
- a keresztsvalidációhoz használt foldok számát (`cv=5`).

Majd a 14. sorban található `fit()` metódus minden paraméterkombinációra lefuttatja a vizsgálatot. Ehhez paraméterként egy `surprise.dataset.Dataset` típusú adatkészletet kell biztosítanunk, mely az alkalmazásban a *MovieLens ml-latest-small* (lásd 4.2.2. fejezet) adatkészletből származik. Az 5.3. kód 16. sorában lévő kóddal megkaphatjuk a legjobb *RMSE* (lásd 3.6.1 fejezet) pontszámot és a 17. sorban lévő kóddal az ehhez tartozó paramétereket. Ezeket a későbbiekben felhasználom.

Ugyanezekkel a paraméterekkel lefuttattam a keresést az *MovieLens ml-latest-small* adatkészleten, és az 5.1. fejezetben leírt *TimeBenchmark*-kal lemértem a futtatáshoz szükséges időt, ami *21281.79* másodperct vett igénybe, ami körülbelül **6 órának** felel meg!³ Az 5.1. ábrán látható, hogy nagyságrendileg a processzorból hány százalékot és a RAM-ból hány MB-ot vett igénybe a futtatás.



5.1. ábra. Hiperparaméter optimalizáció által igénybe vett erőforrások

Az 5.3. fejezetben részletesebben ki fogok térni az algoritmusok összehasonlítására, azonban érdekességeképpen lássuk, hogy az SVD vagy az optimalizált hyperparamérekkel ellátott SVD (Tuned SVD) teljesít jobban az 5.2 ábrán.

	MAE	MSE	RMSE	Hit-Rate	Novelty	Eltel idő (másodpercben)
SVD	0,6723553189	0,7657886275	0,8750934965	0,02295081967	22,38607474	56,42734003
Tuned SVD	0,6688193835	0,7599591937	0,8717563844	0,02459016393	21,86371906	56,46489954
Melyik a jobb?	Tuned SVD	Tuned SVD	Tuned SVD	Tuned SVD	SVD	SVD
Mennyivel?	0,003535935462	0,005829433819	0,003337112069	0,001639344262	0,5223556733	0,03755950928

5.2. ábra. SVD és Tuned SVD összehasonlítása

Nos, ahogy az 5.2. ábra mutatja, a hiperparamérekkel ellátott SVD (Tuned SVD) jobban teljesített, noha nem sokkal. Viszont érdemes figyelembe venni, hogy a használt adatkészlet megközelítőleg minden összes 600 felhasználó 9.000 filmre való 100.000 explicit értékelését tartalmazza.⁴ Ez közel sem mondható *big datának* (big datáról a 2.3.

³ Ezt egy *i7-7700HQ* 2.8 Ghz processzorral és egy *16 GB DDR4 SDRAM*-mal ellátott laptopon futtattam.

⁴ Forrás: <https://grouplens.org/datasets/movielens/>

fejezetben találhatunk információkat). Lehet, a *Netflix* vagy *IMDb* adatbázisával ez a különbség sokkal számottevőbb lenne, noha jóval több mint 6 órába telne a futtatás.

5.3. Algoritmusok összehasonlítása

Ebben a részben az 1.5.3. fejezetben említett *Surprise* könyvtárat használva kerülnek összehasonlításra az alábbi algoritmusok:

- *random_pred.NormalPredictor*: Az algoritmus a tanítókészlet eloszlásán alapuló véletlenszerű értékelés előrejelzésére.
- *knns.KNNBasic*: Egy alap kollaboratív szűrési algoritmus.
- *matrix_factorization.SVD*: A 3.4.3. fejezetben említett SVD algoritmus.
- *matrix_factorization.SVD finomhangolt hiperparaméterekkel*: Az 5.2. fejezetben kapott hiperparamétereket tápláltuk be az SVD algoritmusba.
- *matrix_factorization.SVDpp*: Az *SVD++* algoritmus az *SVD* kiterjesztése, mely figyelembe veszi az implicit értékeléseket.

Ezek mindegyike a *Surprise prediction_algorithms* csomagjában vannak implementálva.[9]

A 3.6. fejezetben tárgyalt mérőszámokból párat fogunk vizsgálni, mégpedig:

- *MAE*
- *MSE*
- *RMSE*
- *Hit-Rate* (Találati arány)
- *Novelty* (Újdonság).

Ezek közül a *MAE*, *MSE* és az *RMSE* implementálva van a *Surprise* könyvtár *accuracy* moduljában.⁵ A *Novelty*-hez pedig egy *recmetrics* nevű GitHub repositoryt valamint *Claude Shannon* információtartalomra vonatkozó definícióját [81] alkalmazom.⁶

Ennek megvalósításához létrehoztam egy *metrics* csomagot (package) az intelligens szolgáltatásban, ezen belül két modul található:

- *prediction_evaluation*: Ebben egy *Metrics* osztály van, amiben implementáljuk a metrikák kiszámítását a paraméterként átadott becslésekből.

⁵ Surprise accuracy moduljának dokumentációja: <https://surprise.readthedocs.io/en/stable/accuracy.html>

⁶ recmetrics novelty forráskódja: <https://github.com/statisticianinstilettos/recmetrics/blob/master/recmetrics/metrics.py#L15>

- *algorithm_evaluation*: Itt egy *AlgorithmEvaluator* osztály található, aminek feladata, hogy a biztosított algoritmusoknak a metrikáit kiértékelje a *Metrics* osztály használatával. Ehhez elvégzi a szükséges adatok előkészítését és az eredmények kiíratásáról is gondoskodik.

5.3.1. Újdonság (novelty) értékének kiszámítása

Az 5.4 kódban látható egy kódrészlet az újdonság értékének kiszámításáról.

```

1 import numpy as np
2
3 class Metrics:
4     # ...
5     @staticmethod
6     def novelty(recommendations, popularity_rankings):
7         :
8         number_of_recommendations = len(next(iter(
9             recommendations.values())))
10        total_occurrences_of_all_items = sum(
11            popularity_rankings.values())
12        mean_self_information = []
13
14        for userID, sublist in recommendations.items():
15            self_information = 0
16            for itemID in sublist:
17                item_popularity = popularity_rankings.get(
18                    itemID, 1) /
19                    total_occurrences_of_all_items
20                self_information += -np.log2(
21                    item_popularity)
22            mean_self_information.append(self_information
23                / number_of_recommendations)
24
25        system_level_novelty = np.mean(
26            mean_self_information)
27        return system_level_novelty,
28            mean_self_information

```

5.4. kód. Újdonság értékének kiszámítása

Ahogy az 5.4. kód 6. sora mutatja, ez a statikus metódus két paramétert kér:

- *recommendations*: ez az algoritmus által generált Top-N ajánlási lista, egy *defaultdict*⁷, melynél a kulcsok a felhasználó ID-k, az értékek *listák* melyek elemei az

⁷A *defaultdict* egy szótár szerű adattípus, bővebben <https://docs.python.org/3/library/collections.html#collections.defaultdict>

ajánlott filmek ID-jából és a becsült értékelésekből készült *tuple*⁸-ök.

- *popularity_rankings*: ez egy úgynevezett *népszerűségi sorrend*⁹, egy *defaultdict*, ahol a kulcsok film ID-k, az értéke pedig az előfordulásának a száma.

Az algoritmus a belső *for* ciklusban (lásd 5.4. kód, 11-15. sorok) számítja ki az elemek információtartalmát, azaz azt, hogy egy adott elem előfordulása mennyire tekinthető valószínűtlennek (minél nagyobb az érték, annál valószínűtlenebb és váratlannabb az elem bekövetkezése). Ehhez Claude Shannon információtartalomra vonatkozó definícióját alkalmazom.[81]¹⁰ Ezt az elemenkénti információtartalmat az 5.4. kód 15. sorában normalizálja a teljes elemszámot felhasználva. Majd a 18. sorban kiszámolja a rendszerre vonatkozó újdonságot, melyet az elemek átlagolásával kap meg. Ez a metódus, a 19. sorban két értéket ad vissza:

- *system_level_novelty*: A teljes rendszerre vonatkozó információtartalmat.
- *mean_self_information*: Egy listát az elemenkénti információtartalommal.

5.3.2. Találati arány (hit-rate) kiszámítása

Az 5.5 kódban látható egy kódrészlet az újdonság értékének kiszámításáról.

```
1 class Metrics:  
2     # ...  
3     @staticmethod  
4     def hit_rate(top_n_recommendations,  
5                   predictions_for_left_out_items) -> float:  
6         num_of_hits = 0  
7         total = len(predictions_for_left_out_items)  
8  
8         if(total < 1):  
9             return None  
10  
11        for user_id, left_out_movie_id, r_ui, est, _ in  
12            predictions_for_left_out_items:  
13                predicted_movies = [movie_id for movie_id, _  
14                    in top_n_recommendations[user_id]]  
15                if left_out_movie_id in predicted_movies:  
16                    num_of_hits += 1
```

⁸ A *tuple* egy vesszővel elválasztott értékekből álló adattípus (például (1, "John Smith")), bővebben: <https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences>

⁹ A népszerűségi sorrendet többféle képpen ki lehet számolni, ez a legegyszerűbb módja.

¹⁰ Az információelméletben az információtartalom, öninformáció, meglepetés vagy Shannon-információ egy alapvető mennyiség, amely egy véletlen változóból származó különleges esemény bekövetkezésének valószínűségből származik.

```
16     return num_of_hits / total
```

5.5. kód. Találati arány kiszámítása

A 3.6.1. fejezetben már írtam a találati arány kiszámításáról, összefoglalva: a felhasználók által 1-1 preferált elemet kiválasztunk, ez lesz a tesztkészletünk, és megnézzük, hogy a rendszer által tett ajánlásokban milyen gyakran fordulnak elő ezek az elemek. Ugyan különböző módszerek vannak ennek kiszámítására, az egyszerűsége miatt azonban erre esett a választásom.

Ahogy az 5.5. kód 4. sorában látható, ennek a metódusnak is két paramétert kell szolgáltatnunk, egy top-n ajánlást (*top_n_recommendations*) és a tesztkészletet (*LOOCV_testset*)¹¹. A 8. sorban ellenőrzi, hogy érvényes-e a teszkészletünk, ha nem, a 9. sorban egy nem informatív *None* értéket ad vissza, de itt hibát is dobhatnánk. A *for* ciklusban, az 5.5. kód 11-14. soraiban számolja a találatokat. Ehhez végigiterál a tesztkészleten, majd az iterációban lévő felhasználó ID-ját (*user_id*) és kihagyott film ID-ját (*left_out_movie_id*), valamint a 12. sorban, a felhasználó ID-ját használva kiszelektálja a felhasználóhoz tartozó ajánlásokat (*predicted_movies*). Ezt használva, a 13. sorban ellenőrzi, hogy a kihagyott film ID-ja szerepel-e az ajánlások között, ha igen, a 14. sorban a számlálóhoz (*num_of_hits*) ad egyet. A metódus végén, a 16. sorban a kapott találatok számát elosztja a tesztkészlet elemeinek számával, és az így megkapott értéket adja vissza, mint találati arány.

5.3.3. A kiértékelés

A kiértékeléshez létrehoztam az említett *algorithm_evaluation* modult, benne az *AlgorithmEvaluator* osztálytal, ami a megtalálható az *./intelligent-service/metrics/algorithm_evaluator.py* elérési úton.

```
1 import pandas as pd
2 from collections import defaultdict
3 from data_handler import DataHandler
4 from metrics.prediction_evaluation import Metrics
5 from utils.top_n import get_top_n
6 from utils.benchmark import TimeBenchmark
7
8 class AlgorithmEvaluator:
9     def __init__(self, data_handler: DataHandler):
10         self.data_handler = data_handler
11         self.time_benchmark = TimeBenchmark()
12
13     def evaluate_algorithms(self, algorithms, top_n=
True, n=10, print_result=True, save_results=
```

¹¹ LOOCV testset: a LOOCV a Leave-One-Out-Cross-Validation rövidítése, és ez annak a tesztkészlete, azaz a felhasználókhöz társított 1-1 elem.

```

        True, save_to="comparing-algorithms.csv"):
14    # ...
15
16    def print_metrics(self, metrics):
17        # ...
18
19    def save_metrics_to_csv(self, metrics, path):
20        # ...
21
22    def __prepare_metrics_df(self, metrics):
23        # ...

```

5.6. kód. AlgorithmEvaluator osztály felépítése

Az 5.6. kódban látható, hogy hogyan is épül fel ez az *AlgorithmEvaluator* osztály. Példányosításkor biztosítanunk kell számára egy *DataHandler* példányt, ami majd a szükséges adatokat fogja számára szolgáltatni, valamint a 11. sorban a konstruktorban létrehoz magának egy *TimeBenchmark* példányt, amit a *time_benchmark* mezőjében tárol, ezzel foglyuk az eltelt időt mérni.

Az osztály tartalmaz 3 publikus

- *evaluate_algorithms*: az paraméterként kapott algoritmusoknak végzi el a kiértékelését, kiíratja és CSV formátumban menti,
- *print_metrics*: kiírja a paraméterként kapott metrikákat ,
- *save_metrics_to_csv*: CSV formátumban menti a paraméterként kapott metrikákat a biztosított elérési útra

és egy privát *__prepare_metrics* metódust mely egy segéd funkciót szolgál, a paraméterként kapott metrikákból csinál egy *pandas.DataFrame*-et.

```

1 import pandas as pd
2 from collections import defaultdict
3 from data_handler import DataHandler
4 from metrics.prediction_evaluation import Metrics
5 from utils.top_n import get_top_n
6 from utils.benchmark import TimeBenchmark
7
8 class AlgorithmEvaluator:
9     # ...
10    def evaluate_algorithms(self, algorithms, top_n=
11        True, n=10, print_result=True, save_results=
12        True, save_to="comparing-algorithms.csv"):
13        metrics = defaultdict(dict)
14        for algo in algorithms:
15            algo_name = algo["name"]

```

```

14     algorithm = algo["algorithm"]
15
16     self.time_benchmark.start()
17     algorithm.fit(self.data_handler.get_trainset()
18                   ())
19     predictions = algorithm.test(self.
20                                   data_handler.get_testset())
21
22     mae = Metrics.mean_absolute_error(predictions
23                                         )
24     mse = Metrics.mean_squared_error(predictions)
25     rmse = Metrics.root_mean_squared_error(
26         predictions)
27     metrics[algo_name] = {"MAE": mae, "MSE": mse,
28                           "RMSE": rmse}
29
30     if top_n:
31         algorithm.fit(self.data_handler.
32                         get_leave_one_out_trainset())
33         predictions_for_left_out_items = algorithm.
34             test(self.data_handler.
35                 get_leave_one_out_testset())
36         predictions_for_anti_testset = algorithm.
37             test(self.data_handler.
38                 get_leave_one_out_anti_testset())
39
40         top_n_predictions = get_top_n(
41             predictions_for_anti_testset, n)
42
43         hit_rate = Metrics.hit_rate(
44             top_n_predictions,
45             predictions_for_left_out_items)
46         system_novelty, _ = Metrics.Novelty(
47             top_n_predictions, self.data_handler.
48             get_popularity_rankings())
49
50         metrics[algo_name]["Hit-Rate"] = hit_rate
51         metrics[algo_name]["Novelty"] =
52             system_novelty
53
54         self.time_benchmark.stop()
55         metrics[algo_name]["Time Elapsed (seconds)"]
56             = self.time_benchmark.elapsed_time
57         metrics[algo_name]["Time Elapsed (minutes)"]
58             = self.time_benchmark.elapsed_time / 60
59
60     if print_result:

```

```

43     self.print_metrics(metrics)
44     if save_results:
45         self.save_metrics_to_csv(metrics, save_to)
46     return metrics
47
48 # ...

```

5.7. kód. EvaluateAlgorithms metódus

Az 5.7. kód 10. sorában látható az *evaluate_algorithms* metódus, a következő paramétereket várja:

- *algorithms*: Egy szótár, amely tartalmazza a vizsgálni kívánt algoritmusokat és azok nevét.
- *top_n*: Egy logikai érték, amelynek alapértelmezett értéke igaz; ha igaz, elvégzi a *hit-rate* és *novelty* értékek kiszámítását és az ehhez szükséges műveleteket.
- *n*: A *top-n* ajánlás *n* értékét határozza meg, alapértelmezett értéke 10.
- *print_result*: Egy logikai érték, amely ha igaz, kiírja a folyamat végén kapott metrikákat, alapértelmezett értéke igaz.
- *save_results*: Egy logikai érték, amely ha igaz, CSV fájlba írja a folyamat végén kapott metrikákat, alapértelmezett értéke igaz.
- *save_to*: Megadja, hova mentse a CSV fájlt, alapértelmezett értéke *comparing-algorithms.csv*.

Az algoritmusok vizsgálata a 12-40. sorokban lévő *for* iterációban történik. A 17. sorban betanítja az aktuális iterációban lévő modellt a *self.data_handler.get_trainset()* által biztosított adatfelosztásos validáció (3.5.3. fejezet) tanító készletével. Majd a 18. sorban a már betanított algoritmustól készített egy ajánlókészletet a *self.data_handler.get_testset()* által kapott, a tanítókészlethez tartozó tesztkészletet felhasználva. A *MAE*, *MSE*, *RMSE* kiértékeléséhez az egyszerűség és a gyorsaság végett használok adatfelosztásos validációt pl. keresztpályázat helyett, melyeket a 20-22. sorokban ki is értékeljük ezeket, a már említett *metrics.prediction_evaluation* modult használva. A 23. soron ezeket meg is adja majd végeredményként szolgáló *metrics* változóban. Az 5.7. kód 25-36. soraiban történik a *Novelty* és *Hit-Rate* értékek kiszámítása. Ehhez a 26. soron látható módon, az algoritmusunkat most a *self.data_handler.get_leave_one_out_trainset()* hívásával kapott *Leave-One-Out-Cross-Validation* (LOOCV) tanító készletével tanítja be. Az *LOOCV* egyrészt elengedhetetlen a *Hit-Rate* kiszámításához, másrészt maximalizáljuk a tanítókészletet, mivel felhasználónként csak egy elemet hagyunk ki, ugyanakkor fennáll a veszélye a *túllieszkedésnek* (lásd 3.5.1. fejezet). A 27. sorban a

test() metódussal az *LOOCV* tesztkészletét használva kiértékeli az algoritmus teljesítményét, úgy hogy előrejelzéseket készít a *tesztkészletben* lévő minden felhasználó-film párosra és visszaad egy listát mely tartalmazza a tényleges és becsült értékeket is. A 28. sorban nagyjából ugyanezt hajtja végre, kivéve, hogy a *self.data_handler.get_leave_one_out_anti_testset()* egy olyan adathalmazt, úgynevezett *anti-testset*-et (a *surprise* könyvtár így hivatkozik erre az adatkészletre) ad vissza, amely tartalmazza az összes lehetséges felhasználó-film párost, amelyek nem szerepelnek az eredeti értékelési adatkészletben. Ez azt jelenti, hogy ezek azok a felhasználó-elem párosok, amelyekre a felhasználó által még nem történt interakció, értékelés. Ez azért fontos, mert éles környezetben ilyen elemekre szeretnénk ajánlásokat készíteni (nem feltétlen szeretnénk eladni valakinek újra és újra ugyanazokat a termékeket, például könyvek esetén). A 30. sorban ezt az adatkészletet felhasználva választja ki a felhasználók számára a legjobbra becsült n (alapértelmezett értéke 10) elemet. A 32. sorban elvégzi a *hit-rate* kiszámítását, majd a 33. sorban az *újdonságét*. Az 5.8. kódban látható mindennek a felhasználása.

```

1  from surprise import SVD, NormalPredictor, KNNBasic
2      , SVDpp
3  from metrics.algorithms_evaluator import
4      AlgorithmEvaluator
5
6  # ...
7  evaluator = AlgorithmEvaluator(data_handler)
8  tuned_params = {
9      'n_factors': 150,
10     'n_epochs': 20,
11     'lr_all': 0.007,
12     'reg_all': 0.02,
13     'lr_bu': 0.005,
14     'lr_bi': 0.005,
15     'reg_bu': 0.05,
16     'reg_bi': 0.05,
17     'reg_pu': 0.05,
18     'reg_qi': 0.05
19 }
20
21 algorithms = [
22     {"algorithm": NormalPredictor(), "name": "Random"
23     },
24     {"algorithm": KNNBasic(), "name": "KNNBasic"}, 
25     {"algorithm": SVD(), "name": "SVD"}, 
26     {"algorithm": SVD(
27         n_factors = tuned_params["n_factors"],
28         n_epochs = tuned_params["n_epochs"],
29         lr_all = tuned_params["lr_all"],
```

```

27     reg_all = tuned_params["reg_all"],
28     lr_bu = tuned_params["lr_bu"],
29     lr_bi = tuned_params["lr_bi"],
30     reg_bu = tuned_params["reg_bu"],
31     reg_bi = tuned_params["reg_bi"],
32     reg_pu = tuned_params["reg_pu"],
33     reg_qi = tuned_params["reg_qi"]),
34     "name": "Tuned SVD",
35   {"algorithm": SVDpp(), "name": "SVDpp"}
36 ]
37 metrics = evaluator.evaluate_algorithms(algorithms)

```

5.8. kód. Algoritmusok kiértékelése

Az 5.8. kód 5. sorában példányosítjuk az *AlgorithmEvaluator* osztályt, majd a hiperparaméter optimalizálásból (lásd 5.2. fejezet) származó adatokat paramétereket láthatjuk a 6-17. sorokban, melyeket majd egy *SVD* algoritmusnak meg is adunk a 23-33. sorokban. A 19-35. sorokban látható, egy lista, mely elemei egy-egy szótár az algoritmusokkal és azok megnevezéseivel. Végül ezzel az *algorithms* változóval meghívjük az *evaluator.evaluate_algorithms* metódusát a 37. sorban. Ennek eredményét láthatjuk az 5.3. ábrán.¹² Ezt a metrikát felhasználva készítettem még másik két ábrát:

- Az 5.4. ábrán látható a leggyorsabban és leglassabban lefutott algoritmusok
- Az 5.5. ábrán látható az egyes vizsgálatokon legjobban teljesített algoritmusok.

Algoritmus	MAE	MSE	RMSE	Hit-Rate	Novelty	Eltelt idő (másodpercben)	Eltelt idő (percben)
Random	1,133294883	2,005212547	1,416055277	0,004918032787	25,49095686	63,91615582	1,065269264
KNNBasic	0,7328833648	0,913614926	0,9558320595	0	25,49095686	153,1214576	2,552024293
SVD	0,6723553189	0,7657886275	0,8750934965	0,02295081967	22,38607474	56,42734003	0,9404556672
Tuned SVD	0,6688193835	0,7599591937	0,8717563844	0,02459016393	21,86371906	56,46489954	0,941081659
SVDpp	0,664750727	0,7530683067	0,8677950833	0,02459016393	22,06012479	1335,828293	22,26380488

5.3. ábra. Algoritmus metrikák és azok összehasonlítása

Leggyorsabb	Leglassabb	Hány másodperccel gyorsabb?
SVD	SVDpp	-1279,400953

5.4. ábra. Algoritmus metrikák és azok összehasonlítása - Leggyorsabb és leglassabb vizsgálatok

Legkisebb MAE	Legkisebb MSE	Legkisebb RMSE	Legnagyobb Hit-Rate	Legnagyobb Novelty
SVDpp	SVDpp	SVDpp	Tuned SVD	Tuned SVD

5.5. ábra. Algoritmus metrikák és azok összehasonlítása - legjobb értékek

¹²Ezt egy *i7-7700HQ* 2.8 Ghz processzorral és egy *16 GB DDR4 SDRAM*-mal ellátott laptopon futtattam.

Az *SVDpp* az *SVD* egy változata, mely értékesíti az implicit értékeléseket. Noha az általunk választott adatkészletben kizárolag explicit értékelések vannak, mégis az *SVDpp* kapta a legjobb értékeléseket, még ha nem is sokkal.

5.4. Adatok tárolása és manipulálása

Az alkalmazáshoz *MovieLens ml-latest-small* (lásd 4.2.2. fejezet) adatkészletét használom, mely eredetileg *CSV*¹³ formátumban vannak tárolva (például lásd az 5.9. kódot).

```
1 userId, movieId, rating, timestamp  
2 1, 1, 4.0, 964982703  
3 1, 3, 4.0, 964981247
```

5.9. kód. ratings.csv első sorai

Az intelligens szolgáltatás fejlesztése elején, ezeket a *CSV* fájlokat használtam adatforrásként, de később, hogy az adatok a *backend szolgáltatás* rendelkezésére is elérhetők legyenek, valamint hogy demonstráljam egy általános webalkalmazás architektúráját az adatokat *PostgreSQL* adatbázisba migráltam (bővebben a 4.2.2. fejezetben). Mind ez jó példa arra, hogy az adatforrás egy alkalmazás élettartalma alatt változhat, ezt érdemes észben tartani, mikor megtervezzük a szoftverünket.

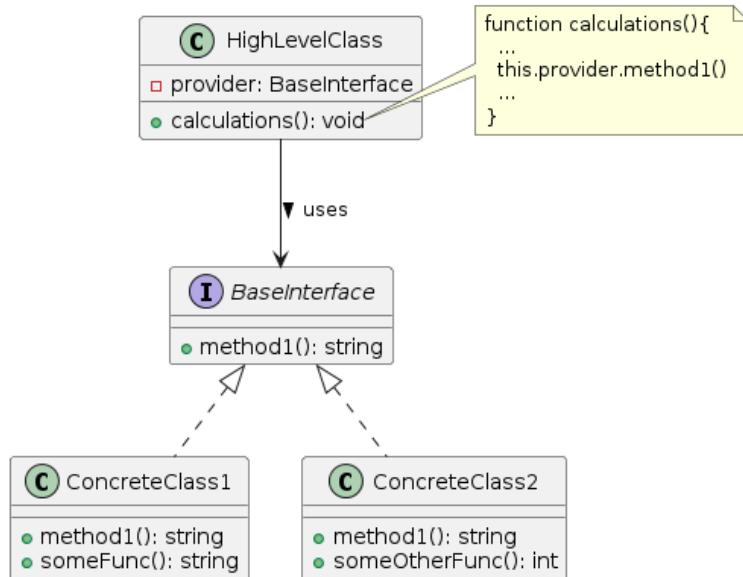
Mindennek a megvalósításához jó hasznát vesszük a *függőség megfordításának elvén* (Dependency Inversion Principle vagy DIP), mely kimondja, hogy:

- A magas szintű kód nem függ az alacsonyabb szintűtől, minden kettő az absztraktiótól függ.
- Az absztraktió nem a részletektől függ, hanem a részletek függnek az absztraktiótól.

[82] Erről látható egy *UML* diagram az 5.6. ábrán:

- A magas szintű kódot a *HighLevelClass* reprezentálja, mely az absztraktiótól, azaz *BaseInterface*-től függ.
- Nem az absztraktió (*BaseInterface*) függ a részletektől (*ConcreteClass1* és *ConcreteClass2*), hanem fordítva.

¹³ CSV: *Comma-Separated-Values* azaz vesszővel elválasztott értékek. Egy szöveges fájl, melyben az adatok vesszővel (vagy más előre definiált karakterrel például pontosvesszővel vannak elválasztva), ezzel biztosítva egy tabuláris adatstruktúrát.



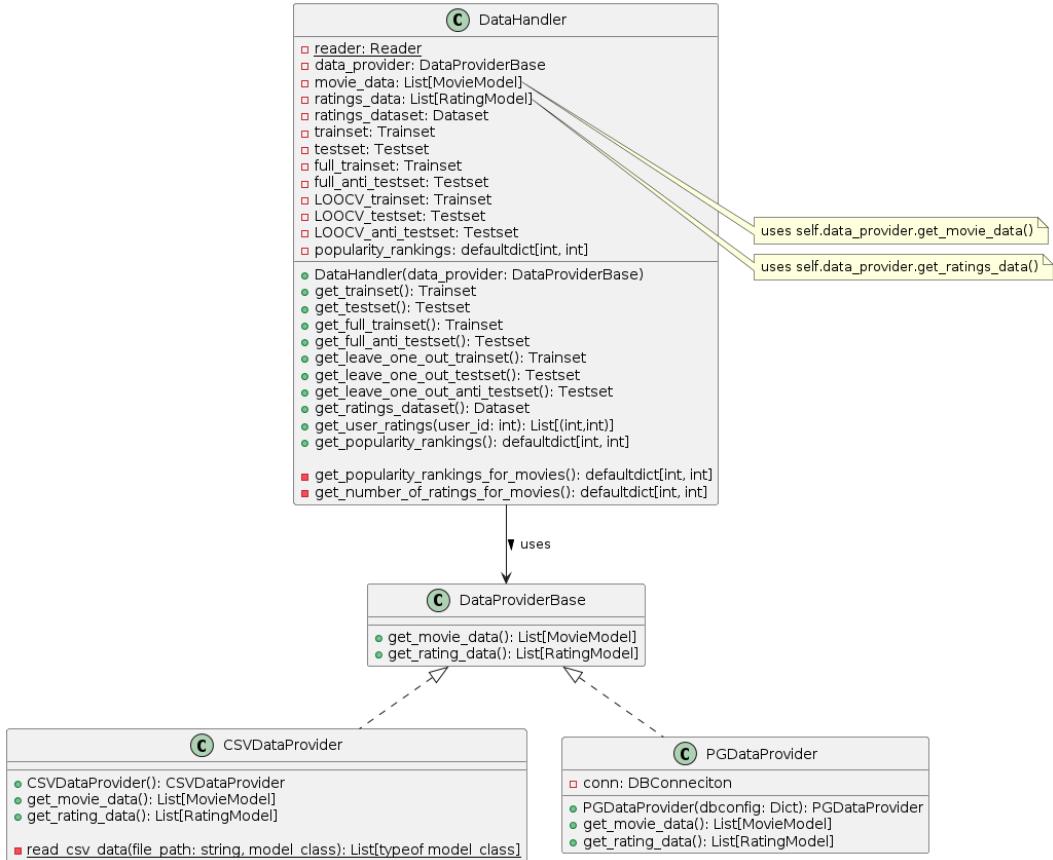
5.6. ábra. Függőség megfordításának elvének bemutatása UML diagramon

5.4.1. Intelligens szolgáltatás

A fent említett függőség megfordításának elvének implementálását láthatjuk az intelligens szolgáltatásban is az 5.7. ábrán látható módon¹⁴. Az intelligens szolgáltatáson belül, az alkalmazáshoz kapcsolódóan minden szükséges adattal kapcsolatos funkcionálitást a *DataHandler* osztályban tárolunk. Egy *függőség befecskendezés* (dependency injection)¹⁵ történik, mikor példányosítjuk a *DataHandler* osztályunkat, ugyanis egy *DataProviderBase* típusú objektumot kell biztosítanunk a konstruktur egy paramétreként. Ezt a *DataProviderBase* objektumot használjuk, mikor a *DataHandler* konstruktőrjük a *movie_data* és *ratings_data* mezők értékét.

¹⁴ A Pythonban nincs a C# nyelvhez hasonló *interface*, ezért ezt egy osztállyal oldom meg.

¹⁵ A *függőség befecskendezése* (dependency injection) a számítógép-programozásban egy technika, aminek lényege, hogy egy objektum más objektumok függőségeit elégíti ki.[83]



5.7. ábra. Függőség megfordításának elvének bemutatása az intelligens szolgáltatásban UML diagramon

Az 5.10. kódban látható, hogy hogyan használhatjuk a különböző `DataProviderBase` implementációit a kódban.

```

1  from data_handler import DataHandler
2  from data_provider.pg_data_provider import
3      PGDataProvider
4  from data_provider.csv_data_prodived import
5      CSVDataProvider
6
7  pg_data_handler = DataHandler(PGDataProvider(
8      db_config))
9  csv_data_handler = DataHandler(CSVDataProvider())
10
11 pg_evaluator = AlgorithmEvaluator(data_handler)
12 csv_evaluator = AlgorithmEvaluator(data_handler)

```

5.10. kód. DataHandler használata különböző DataProviderekkel

5.4.2. Backend szolgáltatás

Architektúra

Ahhoz, hogy megértsük, hogyan történik a *backend szolgáltatás* és az *adatbázis* közötti kommunikáció, nézzük meg egy részletét az 5.11. kódban látható *src*¹⁶ mappa felépítéséről.

```
1  src/
2  |-- api
3  |   '-- v1
4  |       |-- movies
5  |       |   '-- ...
6  |       |-- recommendations
7  |       |   '-- ...
8  |       '-- users
9  |           |-- user-controller.ts
10 |           |-- user-dao.ts
11 |           |-- user-dto.ts
12 |           |-- user-model.ts
13 |           |-- user-routes.ts
14 |           '-- user-service.ts
15 |-- config
16 |   |-- pg-config.ts
17 |   '-- sequelize-config.ts
18 |-- errors
19 |   '-- ...
20 |-- middlewares
21 |   '-- ...
22 |-- models
23 |   '-- ...
24 |-- utils
25 |   '-- ...
26 '-- index.ts
```

5.11. kód. Backend szervíz *src* mappájának tartalmáról egy részlet

A szolgáltatásnál igyekeztem egy *RESTful API*-t létrehozni, ami állapotmentes műveletek sorozatára épít meghatározott erőforrásokkal szemben. Noha a *Richardson Maturity Model*¹⁷-ből az utolsó, 3. szintnek nem teszek eleget, de ez nem is állt szándékban. Továbbá figyelembe vettetem, hogy az API legyen ellátva *verzióval*, és a erőforrások jól elkülöníthetőek legyenek. Így tehát, ahogy az az 5.11. kódban a 2-14. sorokban is látható, azokat az *src/api/v1* mappába rendeztem. Az erőforráson belül a különböző fájlok magyarázata a következő:

¹⁶ A *backend szolgáltatásban* ebben a mappában tároljuk a forráskódokat.

¹⁷ Forrás: <https://martinfowler.com/articles/richardsonMaturityModel.html>

- **-routes.ts*: A beérkező *HTTP(S)* kérést irányítja a megfelelő erőforráshoz, hozzá társítva a kontrollert, és annak funkcióját.
- **-controller.ts*: A kontroller fogadja a felhasználói kéréseket és összekapcsolja azokat a megfelelő szolgáltatásokkal (services) vagy adatkezelő rétegekkel (DAO). Ebben a struktúrában a kontroller a kérés fogadásáért és válasz visszaküldéséért felelős.
- **-service.ts*: A *service* réteg tartalmazza az üzleti logikát és a szükséges adatműveleteket, azaz a kérés feldolgozását végzi.
- **-dao.ts*: A *DAO* (Data Access Object) a konkrét adatelérési logikát kezeli, tehát az *adatbázisműveleteket* hajtja végre.
- **-dto.ts*: Az *adatátviteli objektum* (DTO - Data Transfer Object) lényege, hogy egyszerűsítse és optimalizálja az adatok átvitelét különböző rendszerrések között, minimalizálva ezzel a felesleges adatforgalmat és egyszerűsítve az adatstruktúrákat.[84]
- **-model.ts*: A *Sequelize* használatához szükséges adatmodell definíciót tartalmazzák. Ezek leírják az adatbázis táblák szerkezetét, attribútumait és kapcsolatait, lehetővé téve a *Sequelize* számára, hogy *ORM* (Object-Relational Mapping) módon kezelje az adatbázis-műveleteket.

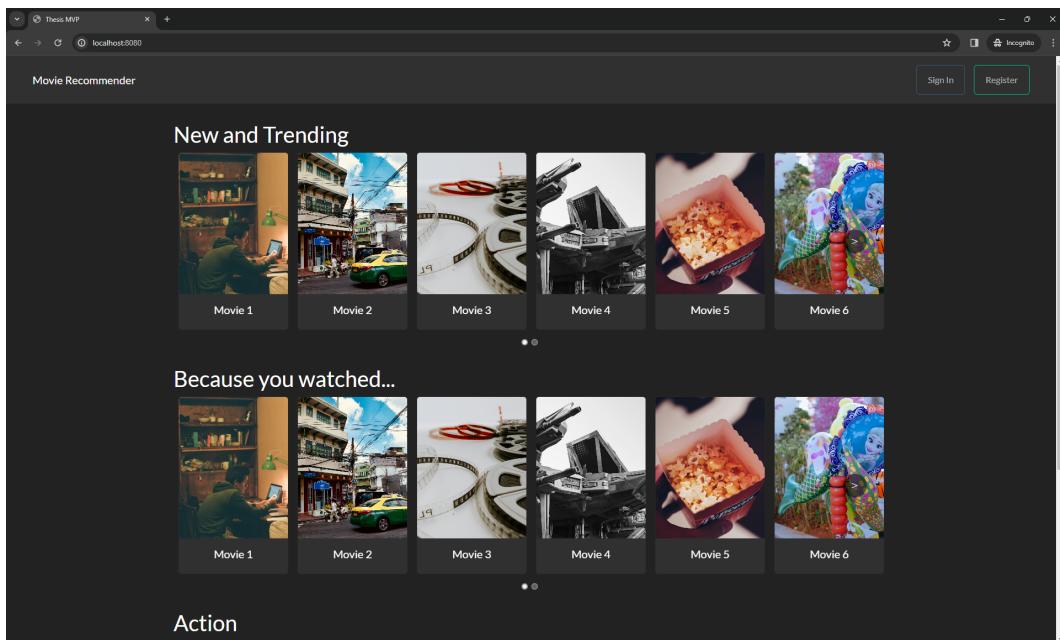
Adatbázissal való kommunikáció

Az adatbázissal való kommunikációhoz a *pg* és a *sequelize* csomagokat használom. Az előbbiit (*pg*) a *PostgreSQL* adatbázishoz való kapcsolódásra míg az utóbbiit (*sequelize*) az *ORM* feladatokat ellátására alkalmazom. A *sequelize* nem csak PostgreSQL, hanem más relációsadatbázis-kezelő rendszerekkel is működik. minden erőforráshoz készítünk egy modellt, mely reprezentálja az adatbázis táblákat, ezt használja majd a *sequelize* csomag. Ez egy kicsit kötöttebb felépítést biztosít, mivel ahhoz hogy más adatbázist használunk, a *./backend-service/src/config/sequelize-config.ts* számára más adatbázis klienst kell biztosítani. Ugyanakkor annak köszönhetően, hogy az 5.4.2. fejezetben bemutatott módon felosztottuk az erőforrásoknak egy-egy feladatát, az így kapott *DAO*-kat már nem kell módosítani. Valamint *DAO*-kat használva a *service* réteg számára már nem számít, honnan származik az adat. Emellett, az egyes erőforrások tudnak kommunikálni egymással is, erre példa, mikor lekérjük a Top-N ajánlási listát a felhasználó számára, az a *RecommendationService*-ból fogja használni a *MovieService*-t, hogy a kapott film ID-khoz lekérje a filmek adatait.

5.5. Az alkalmazás megtekintése

Nézzük meg, hogyan is áll minden össze egy egésszé. Az 5.8. ábrán az alkalmazásunk kezdőlapját láthatjuk, tesztelemekkel feltöltve. Igyekeztem egy *streaming* szolgáltató (pl. Netflix) felületéhez hasonló egyszerűsített kezdőlapot létrehozni. Itt különböző kategóriákat láthatunk, úgy mint:

- „New and Trending”: Új és népszerű filmek. Ezt az adatbázis lekérdezésével a legegyszerűbb meghatározni.
- „Because you watched...”: Amiért megnézted a ... filmet, ajánljuk ezeket a filmeket. Ez egy jó példa a 3.4.1. fejezetben említett *elem-alapú kollaboratív szűrők* felhasználására, például ha a felhasználó többször megnézte *A Mátrixot* (implicit adat), és ötből 5 csillagosra is értékelte (explicit adat), akkor az ehhez hasonló filmeket keressük. De ennek a megvalósítását nem mutatom be a szakdolgozatban.
- „Action”: Különböző műfajokon belüli filmek, ebben a példában akciófilmek. Itt is lehetséges azt megcsinálni, hogy megnézzük a felhasználó által legnézettebb műfajt vagy műfajokat, és azokból ajánlunk más népszerű filmeket, vagy ezt is bővíthetjük még *elem-alapú* szűrőkkel.



5.8. ábra. Az alkalmazás kezdőlapja

5.5.1. Autentikáció

Az 5.8. ábra jobb felső sarkában láthatjuk a *Sign In* (bejelentkezés) és *Register* (regisztráció) gombokat.

A *Register* gomb megnyomása után a felület betölti a regisztrációs űrlapot, mely az 5.9. ábrán látható.

The screenshot shows a dark-themed web page titled "Sign Up". At the top right are "Sign In" and "Register" buttons. Below the title are five input fields: "Name" (placeholder: "Enter name"), "Email Address*" (placeholder: "Enter email"), "Password*" (placeholder: "Enter password"), "Confirm Password" (placeholder: "Enter password again"), and a "Sign Up" button. At the bottom left is a link "Already have an account? [Login](#)". The footer contains the word "Thesis".

5.9. ábra. Regisztrációs felület

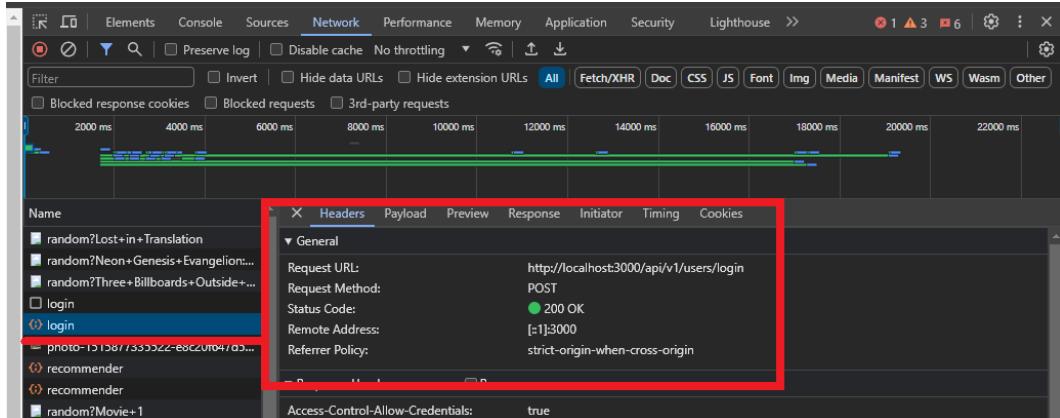
Rendre a *Sign In* gomb megnyomása után a felület betölti a bejelentkezési űrlapot (5.10. ábra).

The screenshot shows a dark-themed web page titled "Sign In". At the top right are "Sign In" and "Register" buttons. Below the title are three input fields: "Email Address*" (placeholder: "Enter email"), "Password*" (placeholder: "Enter password"), and a "Sign In" button. At the bottom left is a link "New user? [Register](#)". The footer contains the word "Thesis".

5.10. ábra. Bejelentkezési felület

A két felület és a folyamat nagyon hasonló. Validációt nem implementáltam, az *Email Address* mezőhöz a *HTML input element type* értékét „email”-re állítottam, valamint a piros csillaggal jelölt mezőknél *required*-ként (kötelezőként) definiáltam az inputokat.

A *Sign In* (bejelentkezés) gomb megnyomását követően az alkalmazás beküldi a *backend szolgáltatás* részére a szükséges adatokat (5.11. és 5.12. ábrák).



5.11. ábra. Bejelentkezési űrlap beküldése a Backend részére

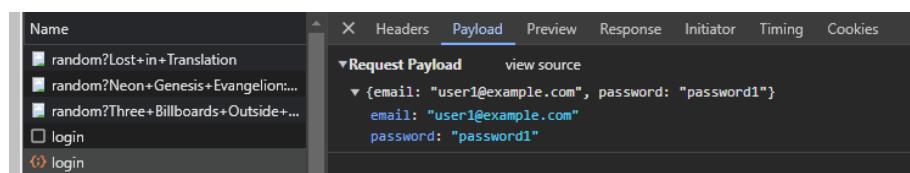
A *backend szolgáltatás* a `./backend-service/src/index.ts` fájlban az `app.use('/api/v1/users', userRouter)`; sorral átirányítjuk a kérést az `userRouter`-nek. Majd az `userRouter`-ben definiált `userRouter.post('/login', UserController.loginUser)` sor meghívja az `UserController.loginUser` statikus metódusát, ami látható az 5.12. kódban.

```

1  public static loginUser = asyncHandler(
2    async (req: Request, res: Response, _next: NextFunction) => {
3      const userDTO = validateRequestBody(req.body);
4      const user = await UserService.loginUser(
5          userDTO);
6
7      if (user && user.id) {
8          setUpJwtCookie(user.id, res);
9          res.status(200).json(user);
10         return;
11     }
12
13     throw new UnauthorizedError();
14 },
15 );

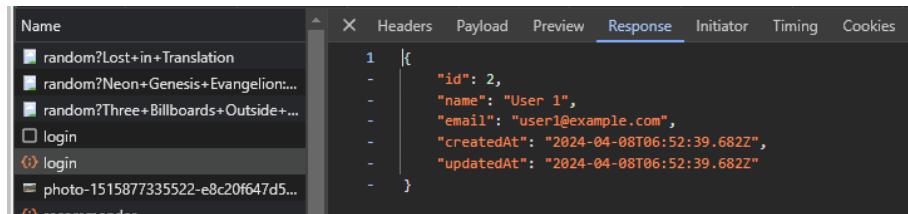
```

5.12. kód. Backend szolgáltatás - UserController.userLogin



5.12. ábra. Bejelentkezési űrlap tartalma

Az 5.12. kód 3. sorában validáljuk a kérésben küldött adatokat, majd a 4. sorban az *UserService*-t használva, az *UserDAO* segítségével megnézzük, hogy létezik-e felhasználó ezzel az *email-jelszó* párossal. Ha a felhasználó megtalálható az adatbázisban és a jelszó is stimmel, válaszként visszaküldjük az adatbázisból származó felhasználó adatokat *JSON formátumban*, hozzá egy 200-as (OK) HTTP kódot társítva, jelezve, hogy minden rendben ment. Ez látható az 5.13. ábrán is. Ellenkező esetben egy *UnauthorizedError*-t dobunk.



5.13. ábra. Backend HTTP válasza a bejelentkezésre

Emellett az 5.12. kód 7. sorában látható, hogy meghívunk egy függvényt, melyet az 5.13. kód mutat részletesebben.

```

1 const cookieName = 'jwt';
2
3 export function setUpJwtCookie(userId: number, res: Response) {
4   if (!process.env.JWT_SECRET) {
5     throw new MissingEnvVarError('JWT_SECRET');
6   }
7
8   const token = jwt.sign({ userId }, process.env.
9     JWT_SECRET, {
10       expiresIn: process.env.NODE_ENV === 'production'
11         ? '1h' : '30d',
12     });
13
14   res.cookie(cookieName, token, {
15     httpOnly: true,
16     secure: process.env.NODE_ENV === 'production',
17     sameSite: 'strict',
18     maxAge: process.env.NODE_ENV === 'production'
19       ? 3_600_000 : 2_592_000_000,
20   });
21 }

```

5.13. kód. Backend szolgáltatás - HttpOnly cookie Beállítása

Az 5.13. kód 8. sorában létrehozunk egy *JWT tokenet* (lásd 4.2.4. fejezet), amit a *protect middleware*-ben (5.14. kód) fogunk használni. Ezzel fogjuk ellenőrizni, hogy érvényes-e a token, azaz, hogy mi a benne található *userId* és hogy lejárt-e. Ezt a

tokent visszaküldjük *HttpOnly* cookieként a 12-16. sorokban látható módon (a cookie látható az 5.14. ábrán) és eltároljuk a *frontenden* melyeket a bejelentkezés után a kliens automatikusan csatolni fog a *HTTP(S)* kérésekhez. Így a *backend* számára a kérésekkel elérhetővé válik a token és tudjuk *autentikálni* a felhasználót, ezáltal *privát útvonalakat* tudunk létrehozni, ahogy az ajánlások útvonala is az.

Name	Value	Dom...	Path	Expir...	Size	Http...	Secu...	Sam...	Parti...	Prior...
jwt	eyJhbGciOiJIUzI1NiIsInR5cI6IkV...eyJ1c2VySWQjOjlsmlhdC16MTcxMjU5OTQxMSwiZ...joNzE1MTIxNDExfQAJvqKy2EsQYGUzloZja1dU3Magkezm7-YMISqF4tLdo	/		10.0 ...	237	✓		Strict		Medi...

5.14. ábra. Bejelentkezsnél beállított Cookie

Így tehát, ha a bejelentkezést követően le szeretnénk kérni a *backendtől* a felhasználónak készített *Top-N* ajánlásokat, a *frontend* ahhoz a kéréshez már automatikusan társítja a kapott cookiet. Mindez az 5.15. ábrán látható. A felső keretben a hívott URL (Request URL), a használt *HTTP metódus* (Request Method) valamint a válasz státusa (Status Code), mely utóbbi 200 OK, azaz sikeres. Alatta a *Request Headers* részben láthatjuk a legalsó keretben lévő *Cookie* mezőt, mely tartalmazza a *backendtől kapott JWT tokent*. Ha hiányzó tokennel vagy cookie-val, azaz bejelentkezés nélkül végeznénk el ugyanezt a hívást, esetleg lejárt vagy invalid a token, akkor *401 Unauthorized* kódot kapnánk.

Request URL:	http://localhost:3000/api/v1/recommender
Request Method:	GET
Status Code:	200 OK
Remote Address:	[::]:3000
Referrer Policy:	strict-origin-when-cross-origin

Cookie:	jwt=eyJhbGciOiJIUzI1NiIsInR5cI6IkV...eyJ1c2VySWQjOjlsmlhdC16MTcxMjU5OTQxMSwiZ...joNzE1MTIxNDExfQAJvqKy2EsQYGUzloZja1dU3Magkezm7-YMISqF4tLdo
---------	---

5.15. ábra. Ajánlások kérése a backend-service API-tól

5.5.2. Személyre szabott ajánlások

Végül, de nem utolsó sorban elérkeztünk a pontra, ahol minden összeáll egy egésszé, és a bejelentkezett felhasználó számára lekérjük a *Top-N* ajánlást.

Ahogy azt az 5.15. ábrán is látható (a bal sávban), bejelentkezést követően egyből le is kérjük a felhasználónak már előre elkészített ajánlásokat (lásd 5.5.2. fejezet). Ehhez a *backend* szolgáltatás *recommendations API*-ját hívja meg a frontend, az 5.5.1. fejezetben tárgyalt *cookie*-val ellátva. Ez a *cookie* tartalmaz egy *JWT token*, melyet használva a *./backend-service/src/middlewares/auth-middleware.ts* fájlban definiált *protect köztes szoftvert* (middleware) – mely az 5.14. kódban látható – alkalmazva *dekódoljuk* és *validáljuk* az ábra 13. sorában, majd az *UserService*-t és a tokenben tárolt *userId*-t használva lekérjük a felhasználó adatait az adatbázisból, ami a 18. sorban olvasható. A 19. sorban a *felügyeleti lánc elvét* (the chain of responsibility)¹⁸ használva, a *request* objektumhoz, melyet a *middleware* tovább ad a következő *middleware*-nek, létrehozunk egy *user* nevű tulajdonságot, és ennek értékül adjuk a Service-től kapott¹⁹ *user*-t. Végül a 20. sorban hívjuk a következő middleware-t.

```
1  export const protect = asyncHandler(
2    async (req: ExtendedRequest, _res: Response, next
3      : NextFunction) => {
4      if (!process.env.JWT_SECRET) {
5          throw new MissingEnvVarError('JWT_SECRET');
6      }
7
7      const token = (req.cookies as { jwt?: string })
8          .jwt;
9      if (!token) {
10          throw new UnauthorizedError('Not authorized,
11              no token');
12      }
13
14      try {
15          const decoded = jwt.verify(token, process.env
16              .JWT_SECRET) as JwtAuthToken;
17          if (!decoded.userId) {
18              throw new Error('Missing field in token');
19          }
20
21          const user = await UserService.getUserById(
22              decoded.userId);
23          req.user = user;
24          next();
25      }
26  }
```

¹⁸The chain of responsibility-ről bővebben: <https://www.geeksforgeeks.org/chain-responsibility-design-pattern/>

¹⁹Az *UserService* az *UserDAO*-t használva szerzi meg az adatbázisban tárolt adatokat.

```

21     } catch (error) {
22         throw new UnauthorizedError();
23     }
24 },
25 );

```

5.14. kód. Backend szolgáltatás - protect middleware

Mivel a hívott API végpont el van látva ezzel a köztes szoftverrel, az ezzel a végponttal összekötött *RecommendationController.getTopNRecommendationForLoggedInUser* statikus metódusában elérhető a *request* objektum *user* tulajdonsága (property), mely *user_id* tulajdonságának értékével meghívjuk a *RecommendationService.getTopNRecommendationsForUserById* (5.15. kód) statikus metódusát és az abból kapott értéket fogjuk visszaküldeni a *frontendnek*.

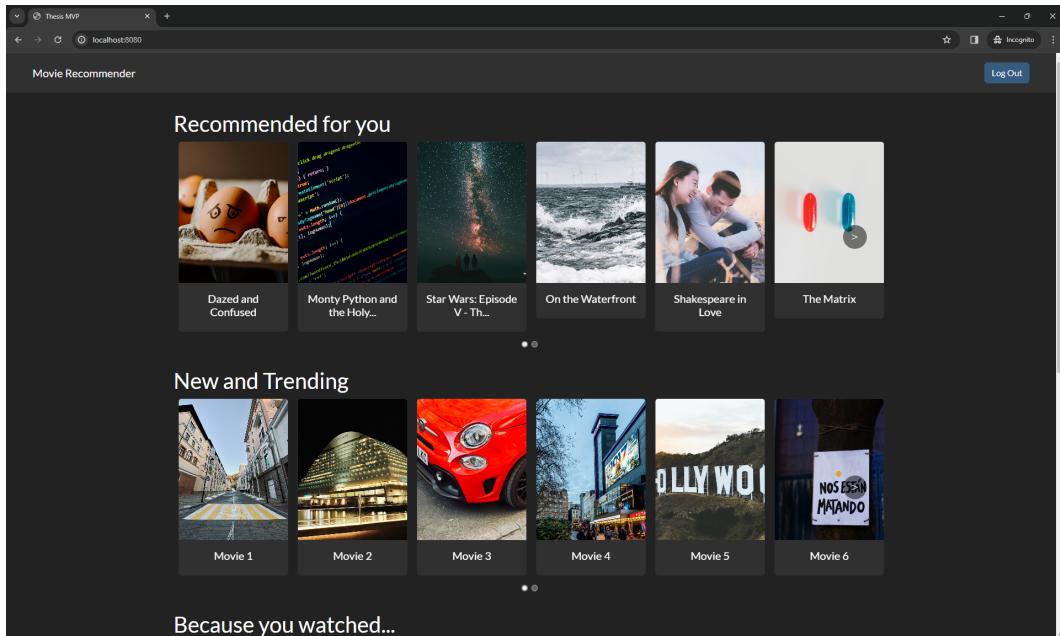
```

1  public static async
2      getTopNRecommendationsForUserById(id: number):
3          Promise<MovieDTO [] > {
4
5      try {
6          const validatedId = validateNumberParam(id);
7          const user = await UserDAO.getUserById(
8              validatedId);
9          if (!user) {
10              throw new NotFoundError(`User with ID ${validatedId} not found`);
11          }
12          const recommendations = await axios.get<
13              RecommendationDTO [] >(
14              `${intelligentServiceRecommendationsURL}/${validatedId}`,
15          );
16          const recIds: number [] = recommendations.data.
17              map((rec) =>
18                  validateNumberParam(rec.movieID),
19              );
20          return MovieService.getMoviesByIdArray(recIds
21      );
22  } catch (err) {
23      if (err instanceof TypeError) {
24          throw new ServiceError(`Provided user id is
25              not a number: ${id}`);
26      }
27      throw err;
28  }
29 }

```

5.15. kód. RecommenderService.getTopNRecommendationsForUserById

Az 5.15. kódban először validáljuk a paraméterként kapott *id*-t majd a felhasználót. Ezután a 8-10. sorokban meghívjuk az *intelligens szolgáltatás* azon végpontját, mely a felhasználó ID-ját használva kikeresi, majd visszaküldi a számára készített ajánlásokat, egy lista formájában, bennük objektumokkal, amik tartalmazzák a *movieID*-t. Ezeket az ID-kat szelektáljuk ki a 11. sorban. Ezt követően a 14. sorban, a *MovieService.getMoviesByArrayOfId*-t hívva megkapjuk a filmek adatait és végül a *backend API* visszaküld egy listát az így kapott filmkről, melyek „azonnal” láthatóak a felhasználó számára a *Recommended for you* listában²⁰, melyet az 5.16. ábra is mutat.



5.16. ábra. Személyre szabott ajánlások megjelenítése a böngészőben

A frontenden ezt az ajánlást az *AppContext*-ból menedzseljük. Az *AppContext*-ból tudunk szolgáltatni funkciókat és állapotokat (state) azoknak a komponensek a számára, melyek ennek a *Context*-nek a *Provider*-jének a gyermekei (ebben az alkalmazásban gyakorlatilag minden komponens számára).²¹ A feladat mérete miatt a *React*-ra jellemző *prop drilling* probléma nincs jelen, azonban az applikáció bizonyos kulcs funkcióit (például autentikáció, ajánlások lekérése) és állapotait (például felhasználó, filmek adatai, hibák) az *AppContext*-tel menedzselem, így központosítva azt. Ezáltal megkapjuk a *HomeScreen* komponens kódját, mely az 5.16. kódban látható.

```

1 import { useEffect } from 'react';
2 import MoviesCarousel from '../components/Movies/
  MovieCarousel';
3 import { useAppContext } from '../app/context/
  AppContext';
4 import { getMockMovies } from '../utils';

```

²⁰ A képek csak reprezentációként szolgálnak

²¹ React useContext hookjáról bővebben: <https://react.dev/reference/react/useContext>

```

5
6  const HomeScreen = () => {
7    const movies = getMockMovies(12);
8    const { appState, getTopN } = useAppContext();
9
10   useEffect(() => {
11     if (appState.user && !appState.movies && !
12       appState.error) {
13       getTopN();
14     }
15   }, [appState.user, appState.movies, getTopN,
16       appState.error]);
17
18   return (
19     <>
20       {appState.movies && (
21         <MoviesCarousel header="Recommended for you"
22           " movies={appState.movies} />
23       )}
24       <MoviesCarousel header="New and Trending"
25           movies={movies} />
26       <MoviesCarousel header="Because you watched
27           ..." movies={movies} />
28       <MoviesCarousel header="Action" movies={movies} />
29     </>
30   );
31
32 };
33
34
35   export default HomeScreen;

```

5.16. kód. HomeScreen komponens

Az 5.16. kód 8. sorában látható, hogy az *AppContext*-et igénybe vesszük, így hozzáférünk annak *appState* és *getTopN* elemeihez. A 10-14. sorokon lévő *useEffect* minden alkalommal lefut, amikor a 14. sorban lévő függőségek – *appState.user*, *appState.movies*, *appState.getTopN*, *appState.error* – megváltoznak. A 11. sorban ellenőrizzük, hogy van-e bejelentkezett felhasználónk (*appState.user*), hogy nincsenek-e még betöltve filmek²² és nincs-e hiba. Ha ezek közül bármelyik hamis, akkor lekérjük a Top-N ajánlási listát a 12. sorban lévő utasítással, mely programkódja az 5.17. kódban látható.

```

1  const getTopN = async () => {
2    const savedRecommendation = localStorage.getItem(
3      'recommendations');
4    try {
5      let topN: MovieDTO[] =

```

²² JavaScriptben az üres tömb ([]) is igaz értéket ad vissza, tehát ![] === false

```

5   if (savedRecommendation) {
6     topN = JSON.parse(savedRecommendation);
7   } else {
8     topN = await Movies.getTopN();
9     localStorage.setItem('recommendations', JSON.
10       stringify(topN));
11
12   dispatch({
13     type: ActionTypes.RECOMMENDATION,
14     movies: topN,
15   });
16 } catch (err) {
17   const error = handleError(err);
18   dispatch({ type: ActionTypes.ERROR, error });
19 }
20

```

5.17. kód. ApplicationContext - getTopN() függvény

Az ajánlásokat a jobb felhasználó élményért *cache*-elem, jelen esetben eltárolom a *local storage*-be. Így amikor meghívom a *getTopN* függvényt, először azt nézem meg, hogy a *local storage*-be vannak-e adatok tárolva (5.17. kód 2-10. sorai), ha vannak, szimplán visszaadom azokat (6. sor), ha nincs, akkor (*axios*-t használva) meghívom a *backend API*-t a 8. sorban látott módon, majd eltárolom a *local storage*-be (9. sor). Ha a folyamat közben hiba lépne fel, azt a 17-18. sorokban látott módon kezeli a program.²³

Modell betanítása, majd annak mentése

Azonban ahhoz, hogy az ajánlások bejárják ezt az utat, az *intelligens szolgáltatástól* egészen a *frontendig*, előtte valahogy létre kell őket hozni. Az 5.3. ábrán láthattuk, hogy az adatmennyiség és az ajánló algoritmus nagyban befolyásolja, mennyi időt lehet igénybe egy-egy algoritmus betanítása. Valamint lehet, hogy más erőforrásokon más módszerekkel szeretnénk betanítani a modelltunkat a hatékonyság vagy költségtakarékosság²⁴ végett. Az 5.17. ábra 2. sorában látható, hogy mennyi időt vesz igénybe egy már betanított modellt és az ahhoz tartozó előre legenerált ajánlásokból létrehozni az összes felhasználó számára egy Top-N ajánlási listát, ahol a $N = 10$.²⁵ Ezzel ellenben, a 3. sorban az látható, hogy mennyi időt vesz igénybe, egy új modellt betanítani ugyanazon az adatkészleten és azzal létrehozni ugyanezt a listát.

²³ A *handleError* függvényre azért van szükség, mert az *Axios* hibáknak más a felépítésük, így azokat külön kell kezelni. Ebben a funkcióban megvizsgálom, hogy *Axios* dobta-e a hibát, ha igen, akkor azt a *handleAxiosError* függvény fogja a továbbiakban lekezelni.

²⁴ Például ha a platform a hívások száma és a futás ideje alapján kalkulálja és számlázza az összeget.

²⁵ Az 5.17. ábrát a *benchmark-loaded-and-fresh-model-svdpp.py* szkripttel hoztam létre.

A modell és az ajánlások mentéséhez valamint annak betöltésére a *surprise.dump* modulját használtam.

	Eltel idő (másodpercben)
Mentett ajánlási lista és algoritmus betöltése	6,640352011
Frissen betanított algoritmus	378,1772263

5.17. ábra. Kész algoritmus és ajánlások betöltése vs. frissen betanított algoritmus²⁶

```

1 import os
2 from flask import Flask, jsonify, make_response
3 from load_dumped import get_top_n
4
5 app = Flask(__name__)
6
7 num_of_top_n = int(os.environ.get(
8     'NUM_OF_TOP_N_ITEMS', 10))
9 rating_threshold = float(os.environ.get(
10    'RATING_THRESHOLD', 4.0))
11
12 @app.route("/api/v1/predictions/<user_id>")
13 def get_top_n_prediction_for_user(user_id):
14     user_ratings = top_n.get(int(user_id), None)
15     if user_ratings is None:
16         return make_response(jsonify({"message": f"No
17             recommendations for user with id: {user_id}"})
18             , 404)
19     return make_response(jsonify(user_ratings), 200)
20
if __name__ == "__main__":
    app.run(debug=True)

```

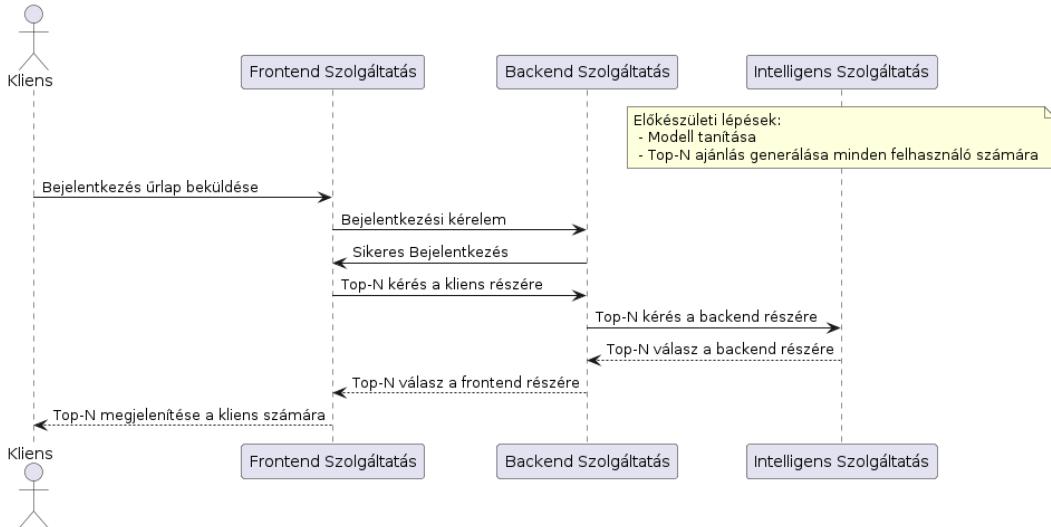
5.18. kód. Intelligens szolgáltatás - *app.py*

Az 5.18. kódban látható minden összes 20 sornyi kód létrehoz egy *API*-t a *Flask* csomagot használva. A kód 10. sorában a *load_dumped.py* fájlban definiált *get_top_n* függvényt használva betölti az összes felhasználó számára generált Top-N ajánlási listát (ahol az *N* a kód 7. sorában van definiálva), ami a *rating_threshold* (a kód 8. sora) érték fölött van. A kód 12-17. soraiban látható a végpont definiálva, melyet a *backend szolgáltatás* hív. Ez lényegében visszaadja a *top_n* változóból az útvonal paraméterben kapott, *user_id* kulcsnak tartozó értéket.²⁷ Majd ezt a kód 17. sorában visszaküldi

²⁶Ezt egy *i5-13600K* 3.5 Ghz processzorral és egy *32 GB DDR5 RAM*-mal ellátott asztali számítógépen futtattam.

²⁷Ha nincs ilyen kulcs, akkor *None*-t.

JSON formátumban. A *backend* ezt továbbítja a *frontend*nek mely megjeleníti azt és a kör bezárul. Ennek a végeredményét láthatjuk az 5.16. ábrán, valamint mindezt szemlélteti az 5.18. ábra is.



5.18. ábra. Szekvenciadiagram

Összegzés

Szakdolgozatomban a felhasznált technológiák rövid bemutatása után a gépi tanulás került terítékre. Kifejtettem, mit is takar ez a kifejezés, valamint ismertettem pár felhasználási területét. Ezt követően a gépi tanulás lépéseiit taglaltam. Majd az adatról esett szó, azzal kapcsolatos kifejezésekről, feladatokról. Ezután a gépi tanulási típusokat és a hiperparaméter optimalizálást boncolgattam. Később az ajánlórendszeret részleteztem, hogy milyen problémákra nyújtanak megoldást, hogyan lehet őket formálisan leírni, valamint különböző hasonlósági függvényeket is fejtegettem. Majd az ajánlórendszer eltérő megközelítéseit vettetem górcső alá, valamint, hogy hogyan készítjük elő az adatot a tanításra, tesztelésre, és hogyan lehet mérhetővé tenni az ajánlórendszer által generált ajánlások pontosságát, hogyan lehet azok különböző tulajdonságait megvizsgálni. Ezt követően definiáltam az alkalmazásom feladatát, specifikáltam annak rendszertervét, architektúráját. Végül bemutattam a szoftver érdekesebb részeit, hogyan közelítettem meg különböző feladatokat, problémákat, és érdekességekkel prezentáltam pár mérési adatot. Betekintést nyújtottam, onnantól, hogy a felhasználó böngészője először megjeleníti az oldalunkat, az autentikáción át a különböző szolgáltatók kommunikációján keresztül az ajánlások generálásával és azok a megjelenítésével bezárólag.

Úgy érzem, a szakdolgozatomban definiált célt kellőképpen teljesítettem és sikerült egy átfogó képet nyújtanom a gépi tanulásról és ajánlórendszerkről, valamint arról, hogyan lehet mindez webalkalmazás formájában megvalósítani. A megírásához és az írása alatt egyaránt, magam is rengeteget tanultam és fejlődtem. Remélem aki olvassa hasznát tudja venni ennek az irománynak és ha netán még örömet is leli az olvasásában, annak különösképpen örülök.

Irodalomjegyzék

- [1] TypeScript (2024.02.09.)
<https://en.wikipedia.org/wiki>TypeScript>
- [2] React (software) (2024.02.09.)
[https://en.wikipedia.org/wiki/React_\(software\)](https://en.wikipedia.org/wiki/React_(software))
- [3] Node.js hivatalos oldala
<https://nodejs.org/en>
- [4] Express.js hivatalos oldala
<https://expressjs.com/>
- [5] REFACTORING GURU: *Chain of Responsibility*
<https://refactoring.guru/design-patterns/chain-of-responsibility>
- [6] SEQUELIZE
<https://sequelize.org/>
- [7] Flask (web framework) (2024.02.17.)
[https://en.wikipedia.org/wiki/Flask_\(web_framework\)](https://en.wikipedia.org/wiki/Flask_(web_framework))
- [8] Surprise könyvtár hivatalos oldala
<https://surpriselib.com/>
- [9] Surprise - prediction_algorithms package documentation (a *Surprise* könyvtár dokumentációjának egy aloldala)
https://surprise.readthedocs.io/en/stable/prediction_algorithms_package.html
- [10] Tune algorithm parameters with GridSearchCV (a *Surprise* könyvtár dokumentációjának egy aloldala)
https://surprise.readthedocs.io/en/stable/getting_started.html#tune-algorithm-parameters-with-gridsearchcv
- [11] Adatbázis (2023.09.19.)
<https://hu.wikipedia.org/wiki/Adatb%C3%A1zis>

- [12] PostgreSQL dokumentációja
<https://www.postgresql.org/docs/current/intro-whatis.html>
- [13] Docker dokumentációja
<https://docs.docker.com/>
- [14] Docker dokumentáció - Docker overview
<https://docs.docker.com/get-started/overview/>
- [15] Docker dokumentáció - What is a container?
<https://docs.docker.com/guides/docker-concepts/the-basics/what-is-a-container/>
- [16] Docker dokumentáció - What is an image?
<https://docs.docker.com/guides/docker-concepts/the-basics/what-is-an-image/>
- [17] Docker Compose dokumentációja
<https://docs.docker.com/compose/>
- [18] Node.js dokumentációja
<https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>
- [19] Machine learning (2024.03.10.)
https://en.wikipedia.org/wiki/Machine_learning
- [20] OLIVER THEOBALD: *Machine Learning For Absolute Beginners: A Plain English Introduction (Second Edition)* (2021)
- [21] Mesterséges Intelligencia (2024.03.12.)
https://hu.wikipedia.org/wiki/Mesters%C3%A9ges_intelligencia
- [22] Artificial Intelligence (2024.03.08.)
https://en.wikipedia.org/wiki/Artificial_intelligence
- [23] The Machine Learning Life Cycle Explained (2020. október)
<https://www.datacamp.com/blog/machine-learning-lifecycle-explained>
- [24] Feature engineering magyarul (2012.05.13.)
<https://angolulblog.wordpress.com/2012/05/13/feature-engineering-magyarul/>
- [25] GARETH JAMES: *An Introduction to Statistical Learning: with Applications in R*, (2013.)

- [26] FARKAS RICHÁRD: *Gépi tanulás alapfogalmai*
<https://www.inf.u-szeged.hu/~rfarkas/ML20/alapfogalmak.html>
- [27] SINDHU V., NIVEDHA S., PRAKASH M.: *An Empirical Science Research on Bioinformatics in Machine Learning.* (2020. Február) *Journal of Mechanics of Continua and Mathematical Sciences*
https://www.journalimcms.org/special_issue/an-empirical-science-research-on-bioinformatics-in-machine-learning
- [28] What is Tabular Data? (Definition & Example), (2022.03.25.)
<https://www.statology.org/tabular-data/>
- [29] Supervised Learning
<https://cloud.google.com/discover/what-is-supervised-learning#section-5>
- [30] A gépi tanulás 3 fő típusa
<https://netliferobotics.hu/blog/a-gepi-tanulas-3-fo-tipusa>
- [31] Unsupervised Learning
<https://cloud.google.com/discover/what-is-unsupervised-learning>
- [32] Klaszteranalízis (2023.03.15.)
<https://hu.wikipedia.org/wiki/Klaszteranal%C3%ADzis>
- [33] OLIVIER CHAPELLE, BERNHARD SCHÖLKOPF, ALEXANDER ZIEN: *Semi-Supervised Learning*, 2006
<https://www.molgen.mpg.de/3659531/MITPress--SemiSupervised-Learning.pdf>
- [34] Mi a gépi tanulás?
<https://www.sap.com/hungary/products/artificial-intelligence/what-is-machine-learning.html>
- [35] What is reinforcement learning?
<https://aws.amazon.com/what-is/reinforcement-learning>
- [36] What is Hyperparameter Tuning?
<https://aws.amazon.com/what-is/hyperparameter-tuning>
- [37] Hyperparameter optimization, (2024.01.04.)
https://en.wikipedia.org/wiki/Hyperparameter_optimization
- [38] Ajánlórendszer (marketing), (2023.04.20.)
[https://hu.wikipedia.org/wiki/Aj%C3%A1nl%C3%A1%C3%B3rendszer_\(marketing\)](https://hu.wikipedia.org/wiki/Aj%C3%A1nl%C3%A1%C3%B3rendszer_(marketing))

- [39] Recommender system (2024.03.11.)
https://en.wikipedia.org/wiki/Recommender_system
- [40] WERNER ÁGNES: *Ajánló rendszerek*, 2019
- [41] Collaborative filtering (2024.02.07.)
https://en.wikipedia.org/wiki/Collaborative_filtering
- [42] Matrix factorization (recommender systems), (2023.12.26.)
[https://en.wikipedia.org/wiki/Matrix_factorization_\(recommender_systems\)](https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems))
- [43] NITIN PRADEEP KUMAR, ZHENZHEN FAN: *Hybrid User-Item Based Collaborative Filtering*, 2015.
<https://www.sciencedirect.com/science/article/pii/S1877050915023492>
- [44] PANG-NING TAN, MICHAEL STEINBACH, VIPIN KUMAR: *Introduction to Data Mining*, 2014.
- [45] XIANGNAN HE, LIZI LIAO, HANWANG ZHANG, LIQIANG NIE, XIA HU, TAT-SENG CHUA: *Neural Collaborative Filtering*. 2017 IEEE Conference on Computer Communications (INFOCOM), 2017.
<https://arxiv.org/abs/1708.05031>.
- [46] CHARU C. AGGARWAL: *Recommender Systems: The Textbook*, 2016.
- [47] PETER BUSILOVSKY: *The Adaptive Web*, 2007.
- [48] Hosszú farok (2024.03.11.)
https://hu.wikipedia.org/w/index.php?title=Hossz%C3%BA_farok
- [49] Cosine similarity (2024.02.15.)
https://en.wikipedia.org/wiki/Cosine_similarity
- [50] Skaláris szorzat (2024.02.10.)
https://hu.wikipedia.org/wiki/Skal%C3%A1ris_szorzat
- [51] Euklideszi norma (2023.07.12.)
https://hu.wikipedia.org/wiki/Euklideszi_norma
- [52] Távolság (2023.12.25.)
<https://hu.wikipedia.org/wiki/T%C3%A1vols%C3%A1g>
- [53] Korrelációs együttható (2022.01.09.)
https://hu.wikipedia.org/wiki/Korrel%C3%A1ci%C3%B3s_egy%C3%BCtthat%C3%A1g

- [54] tf-idf (2024.03.17.)
<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>
- [55] STEPHEN GOWER: *Netflix Prize and SVD*, 2014.04.18.
- [56] Y. KOREN, R. BELL, C. VOLINSKY: *Matrix Factorization Techniques for Recommender Systems*. IEE Computer Society, 2009
- [57] Latent and observable variables (2024.01.26.)
https://en.wikipedia.org/wiki/Latent_and_observable_variables
- [58] KORMÁNYOS ANDOR: *Szinguláris érték felbontás és függvényillesztés*, 2020.02.20.
- [59] Diagonális mátrix (2023.02.18.)
https://hu.wikipedia.org/wiki/Diagon%C3%A1lis_m%C3%A1trix
- [60] Overfitting (2020.11.16.)
<https://lexiq.hu/overfitting>
- [61] Előrejelzések verifikációja II. (2015.04.14.)
https://www.szupercella.hu/elorejelzesek_verifikacioja_2
- [62] Surprise documentation - Accuracy Module
<https://surprise.readthedocs.io/en/stable/accuracy.html#surprise.accuracy.rmse>
- [63] Hit rate (2022.09.29.)
https://en.wikipedia.org/wiki/Hit_rate
- [64] SUSAN LI: *Evaluating A Real-Life Recommender System, Error-Based and Ranking-Based*. 2019.01.17.
<https://towardsdatascience.com/evaluating-a-real-life-recommender-system-error-based-and-ranking-based-84708e3285b>
- [65] RISHABH BHATIA: *Recommendation System Evaluation Metrics*. 2019.03.18.
<https://medium.com/@rishabhhatia315/recommendation-system-evaluation-metrics-3f6739288870>
- [66] 10 metrics to evaluate recommender and ranking systems
<https://www.evidentlyai.com/ranking-metrics/evaluating-recommender-systems#behavioral-metrics>
- [67] How can recommender systems learn from serendipitous discoveries and outcomes?
<https://www.linkedin.com/advice/0/how-can-recommender-systems-learn-from-serendipitous>

- [68] CLAIRE LONGO: *Evaluation Metrics for Recommender Systems* 2018.11.23.
<https://towardsdatascience.com/evaluation-metrics-for-recommender-systems-df56c6611093>
- [69] ZUZANNA DEUTSCHMAN: *Recommender Systems: Machine Learning Metrics and Business Metrics*, 2023.08.07.
<https://neptune.ai/blog/recommender-systems-metrics>
- [70] A/B testing (2024.03.29.)
https://en.wikipedia.org/wiki/A/B_testing
- [71] A/B Testing Guide
<https://vwo.com/ab-testing/#how-to-perform-an-a-b-test>
- [72] LIANG ZHANG: *The Definition of Novelty in Recommendation System*
<https://pdfs.semanticscholar.org/e417/a959f8431a61e9cd74e726ea6367f00a82d6.pdf>
- [73] Systems Design (2024.02.15.)
https://en.wikipedia.org/wiki/Systems_design
- [74] The complete guide to System Design in 2024, (2024.02.15.)
<https://www.educative.io/blog/complete-guide-to-system-design>
- [75] Szolgáltatásorientált architektúra (2023.09.15.)
https://hu.wikipedia.org/wiki/Szolg%C3%A1ltat%C3%A1sorient%C3%A1lt_architekt%C3%BAra
- [76] Az MVP és MMF jelentése, célja, szerepe (2023.11.27.)
<https://promanconsulting.hu/mvp-mmff/>
- [77] Authentication (2024.03.29.)
<https://en.wikipedia.org/wiki/Authentication>
- [78] Közbeékelődéses támadás (2023.11.29.)
https://hu.wikipedia.org/wiki/K%C3%B6zbe%C3%A9kel%C3%B3d%C3%A9ses_t%C3%A1mad%C3%A1s
- [79] KEVIN LIAO: *Prototyping a Recommender System Step by Step Part 2: Alternating Least Square (ALS) Matrix Factorization in Collaborative Filtering*, 2018.11.17.
<https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1>

- [80] Benchmark (computing), (2024.02.20.)
[https://en.wikipedia.org/wiki/Benchmark_\(computing\)](https://en.wikipedia.org/wiki/Benchmark_(computing))
- [81] Information content (2024.03.21.)
https://en.wikipedia.org/wiki/Information_content
- [82] A függőség megfordításának elve (2023.06.16.)
https://hu.wikipedia.org/wiki/A_f%C3%BCgg%C5%91s%C3%A9g_megford%C3%ADt%C3%A1s%C3%A1nak_elve
- [83] A függőség befecskendezése (2023.05.30.)
https://hu.wikipedia.org/wiki/A_f%C3%BCgg%C5%91s%C3%A9g_befecskendez%C3%A9se
- [84] Adatátviteli objektum (2020.06.06.)
https://hu.wikipedia.org/wiki/Adat%C3%A1tviteli_objektum

NYILATKOZAT

Alulírott Besenyei Ferenc....., büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, Ajánlórendszeres és azok integrálása webalkalmazásokban... című szakdolgozat (diplomamunka) önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Tudomásul veszem, hogy a szakdolgozat elektronikus példánya a védés után az Eszterházy Károly Katolikus Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: Eger....., 2024. év április..... hó 15.... nap.

Besenyei Ferenc.....

aláírás