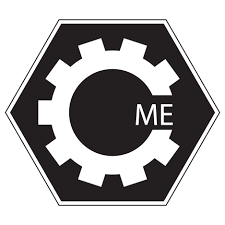
****

***Szakdolgozat***

Siska Dávid

****

**Szakdolgozat**

**Kérdőív alapú adaptív ajánlórendszer fejlesztése**

Készítette: Siska Dávid

Gazdaságinformatikus

Témavezető: Dr. Glavosits Tamás egyetemi adjunktus

**Miskolci Egyetem**

**2024**

**Tartalomjegyzék**

Tartalom

[1. Bevezetés 4](#_Toc180157188)

[2. Ajánlórendszer építése 6](#_Toc180157189)

[2.1 Alternatív keresők bemutatása 6](#_Toc180157190)

[2.1.1 Kulcsszó alapú keresők 6](#_Toc180157191)

[2.1.2 Szemantikus keresők 7](#_Toc180157192)

[2.1.3 NLP (Természetes nyelvfeldolgozás) 7](#_Toc180157193)

[2.1.4 Faceted kereső 8](#_Toc180157194)

[2.1.5 Kép alapú keresők (CBIR) 8](#_Toc180157195)

[2.2 SQLite bemutatása 8](#_Toc180157196)

[2.3 Az adatbázis felépítése 10](#_Toc180157197)

[2.3.1 Az adatbázis szerkezete 10](#_Toc180157198)

[2.3.2 Az adatbázis ER modellje 14](#_Toc180157199)

[2.3.3 Az adatbázis relációs modellje 15](#_Toc180157200)

[2.4 Kézzel készített döntési fa a film nyelvére vonatkozóan 17](#_Toc180157201)

[2.5 Kézzel készített döntési fa hatékonysága 18](#_Toc180157202)

[2.6 Folyamatábra az adatbázis adatainak feldolgozásáról 20](#_Toc180157203)

[2.7 ML.NET bemutatása 20](#_Toc180157204)

[2.7.1 A mesterséges intelligencia és a gépi tanulás kapcsolata 20](#_Toc180157205)

[2.7.2 Mi az az ML.NET? 21](#_Toc180157206)

[2.8 Hogyan néz ki egy ML.NET alkalmazás? 22](#_Toc180157207)

[2.8.1 MLContext létrehozása 22](#_Toc180157208)

[2.8.2 Adatok betöltése 22](#_Toc180157209)

[2.8.3 Adatok átalakítása 23](#_Toc180157210)

[2.8.4 Algoritmus kiválasztása 24](#_Toc180157211)

[2.8.5 Modell tanítása 25](#_Toc180157212)

[2.8.6 Modell kiértékelése 26](#_Toc180157213)

[2.8.7 A modell kimentése, használata 28](#_Toc180157214)

[2.9 Algoritmusok hatékonyságának összehasonlítása 29](#_Toc180157215)

[5. Irodalomjegyzék 78](#_Toc180157216)

# 1. Bevezetés

Napjainkban az embereknek számos döntést kell meghozniuk nap mint nap, melyek sokszor rendkívül fontosak, amelyek nagy átgondoltságot, hosszas gondolkodást igényelhetnek, hiszen hosszan tartó, meghatározó következményeik lehetnek. Máskor kisebb hangsúlyú, kisebb jelentőséggel bíró döntéseket kell meghozni, mint például egy-egy termék megvásárlása, viszont ezek is az emberek feladatai közé tartoznak, és nehéz dolguk lehet a választásban.

Bármilyen döntésről is legyen szó, az emberek mindig próbálnak mérlegelni, figyelembevenni az egyes döntéseknek a lehetséges következményeit, tehát észérvek alapján meghozni egy-egy választást, viszont ez sokszor nem egyszerű, és nem is sikerül. Az emberek sokszor nem rendelkeznek kellő információval és háttértudással egy-egy tématerületet illetően ahhoz, hogy ezeket a tématerületeket érintő kérdésekben logikusan tudjanak dönteni. Ilyenkor a kellő háttértudás hiánya miatt az érzelmeik, „megérzéseik” döntenek egy-egy döntés meghozatala során, és ezáltal előfordulhat, hogy nem a leglogikusabb, legcélszerűbb döntést hozzák meg a lehetséges opciók közül.

A gépek, programok alapvetően nem rendelkeznek semmilyen témát illetően semmilyen tudással. Viszont ha az adott tématerületet illetően a logikus döntéshozáshoz szükséges adatokat megkapják, és emellé párosul egy olyan algoritmus, amely a kellő alapossággal és módszerekkel képes elemezni, feldolgozni ezeket az adatokat, akkor egy olyan mögöttes összefüggéseket figyelembe vevő megközelítés alapján tudnak döntést hozni [1], amelyet a felhasználók a felszíni tulajdonságok vizsgálatával észre sem vennének.

A gépi tanulás egy altípusa a mélytanulás, amelyet pedig reprezentációs tanuláshoz használnak. [2] Ennek célja, hogy nagy mennyiségű jelöletlen adatból, amelyek lehetnek képek, hangok, szövegek, olyan őket reprezentáló beágyazási vektorokat hozzanak létre, amelyek nem a felszíni tulajdonságaik alapján írják le az egyedeket, hanem a mélyebb, mögöttes tartalmi összefüggésekre helyezik a hangsúlyt. [1]

Az ajánlórendszer kifejezést olyan eszközök, szoftverek összességére használjuk, amelyek értékes információkat tudnak szolgáltatni a felhasználók számára abban a formában, hogy mindennapi felmerülő kérdésekre próbálnak választ adni egy-egy döntés formájában. Ezt olyan módon teszik, hogy a felhasználó igényeinek megfelelő dolgokat, termékeket ajánlanak neki. A mai világban, ahol egyre több-és több termék elérhető és rendelhető meg az interneten, ez egy rendkívül hasznos funkció tud lenni a felhasználók, vásárlók számára, hiszen egyébként töménytelen mennyiségű választási lehetőségből kellene válogatniuk és választaniuk, ami megbonyolítja az egész döntési folyamatot. A legtöbbször ezeket az ajánlórendszereket a felhasználók nem is tudatosan használják, hiszen az általuk használt weboldalak, webáruházak, boltok online oldalaiba vannak beépítve, mivel a boltoknak is érdekük, hogy a vásárlók minél hamarabb megtalálják a számukra minél tökéletesebb termékeket. [3] Az ajánlórendszereket gyakran ötvözik mélytanulási módszerekkel is annak érdekében, hogy növeljék hatékonyságukat, és különböző felmerülő problémákat küszöböljenek ki [2].

Az ajánlórendszerek működési koncepciójából adódik, hogy minden ajánlórendszernek szüksége van bemeneti adatokra, amelyekkel a felhasználó igényeinek megfelelő javaslatokat tud tenni. A bemeneti adatok a következők lehetnek: [3]

* Adatok, amelyeket a felhasználó jelenléte hoz be a rendszerbe. A felhasználó személyes adatai, mint pl. a neme, életkora, érdeklődési köre, munkaköre stb.
* Az egyedekre vonatkozó jellemzők, amelyek nagyban függenek az egyed jellegétől.
* A felhasználótól kapott visszajelzések, amelyek lehetnek az ajánlórendszerbe tudatosan bevitt adatok, mint pl. értékelések, vagy akár korábbi vásárlások, amelyeket fel lehet használni a jövőben az ajánlásokhoz.

Az ajánlórendszerek alapját az osztályozók adják, ezért fontos, hogy a különböző osztályozók közül melyiket választjuk, hiszen nagyban tudja befolyásolni a rendszer pontosságát. Osztályozáshoz választhatjuk akár a K-Közép, Naiv Bayes, vagy akár az SVM (vektorgépek támogatása) osztályozókat, vagy akár a döntési fákat is. A döntési fák egyszerűen alkalmazhatóak és gyorsak, ezért egy jó választás lehet őket osztályozóként használni az ajánlórendszerekhez. [4]

A döntési fa algoritmusok működésének alapja, hogy szükségük van egy olyan adathalmazra, amely be tudja „tanítani” őket. Tehát ennek az adathalmaznak, amelyet gyakran tanítóhalmaznak is neveznek, az osztályai ismertek, ezek egymást kizáró címkék, kategóriák. Az egyedeket sokszor attribútum-értékvektorokként ábrázolják, amelyeknek ismert az a tulajdonságuk, hogy milyen osztályba tartoznak. Ezekből az egyedekből áll a fa tanítóhalmaza. A döntési fa ennek a halmaznak a segítségével osztályozza a még nem ismert, és az ismert elemeket is, mindkettő esetben helyesen meg kell tudnia becsülni azt, hogy az adott egyed melyik osztályba fog tartozni. Ez akkor kerül meghatározásra, ha a felépített fában fentről lefelé haladva elérünk egy levelet, hiszen az egymást kizáró osztályok levelek formájában jelennek meg a fában. [5] A levelekig a fa győkerétől a csomópontokon keresztül jutunk el, amelyek egy-egy elágazást valósítanak meg. A csomópontok kialakítása az egyedek különböző attribútumai alapján történnek, a tanító adatok halmazát legjobban felosztó attribútum fogja a fa gyökerét alkotni. Ezután a felosztott adatokat is tovább fogjuk osztani, és ezzel elindul egy rekurzív folyamat, amely akkor fejeződik be, ha az adott ágon az összes maradék adat egy osztályba sorolható. [6]

Mint korábban említésre került, az ajánlórendszerek egyik bemeneti adatai a felhasználóra vonatkoznak. [3] Ezeket a felhasználóra vonatkozó adatokat bekérhetjük valós időben, így aktuálisak lesznek, és akár el is tárolhatjuk őket ahhoz, hogy a későbbiekben felhasználjuk őket, hogy minél több adat álljon a rendszernek a rendelkezésére. Másik bemeneti adatai az ajánlórendszereknek az egyedekre vonatkozó információk. [3] Ezeknek az adatoknak egy részét érdemes lehet tárolni valahol a programon kívül annak érdekében, hogy a program futása után is megmaradjanak, illetve lehetnek viszonylag állandó jellegű adatok is, mint pl. amelyek az egyedekre vonatkoznak. Ezeket az információkat érdemes lehet egy adatbázisban tárolni, hiszen nagyméretű adathalmazok is lehetnek a bemeneti adatok. [4]

A szakdolgozatomban egy filmekre vonatkozó ajánlórendszert fejlesztettem C# nyelven, amely filmeket ajánl a felhasználó számára az általa adott bemeneti adatok felhasználásával. A felhasználónak a filmek különböző tulajdonságaira vonatkozó prioritásai alapján teszi fel a kérdéseket, és súlyozza a felhasználó által ezekre a kérdésekre adott válaszokat. A teljes adathalmazból véletlenszerűen meghatározásra kerül egy 20 elemű tanítóhalmazt, amelyben a halmaz minden eleme kap egy lebegőpontos pontszámot, amelyek a kérdésekre adott válaszokkal kerülnek kialakításra. A felhasználó számára fontosabbnak, tehát nagyobb prioritásúnak beállított tulajdonságokhoz kapcsolódó kérdésekre adott válaszok nagyobb mértékben fogják befolyásolni a pontszámokat. Végül az így kialakított tanítóhalmaz pontszámait felhasználva megbecsüljük az egész adathalmaz filmjeinek a pontszámait, és a legjobb 5 filmet fogja a program ajánlani a felhasználó számára.

Az információk a programomhoz használt filmekről a „movies.db” adatbázisban kerültek letárolásra. Az adatbázishoz SQLite [7] formátumot használtam annak egyszerűsége, hordozhatósága miatt, illetve egy nagy előnye, hogy megbízhatóan tud működni gyakorlatilag szinte minden programozási környezetben.

A felhasznált adatok egy kaggle.com-on megtalálható adatbázisból [8] származnak, amely a TMDB [9]-ről származó adatokkal lett feltöltve, így az általam készített adatbázis adatainak az eredeti forrása a TMDB-n szereplő adatok.

# 2. Ajánlórendszer építése

## 2.1 Alternatív keresők bemutatása

Az ajánlórendszereken kívül természetesen vannak egyéb keresők is, mindegyikük más és más elven alapul, és ennek megfelelően más és más célokra is használatosak. A különböző célok mellett ezek a keresők különböző módokon, és helyeken jelenhetnek meg, mint ahogyan az ajánlórendszerek is, hiszen pl. a Youtube-on megtekintett filmek jelentős része ajánlórendszerek ajánlásából származik [2], de emellett felhasználhatóak akár online webáruházak és könyvtárak oldalain is. [3]

### 2.1.1 Kulcsszó alapú keresők

A kulcsszó alapú keresők a felhasználó által megadott kulcsszavak alapján végzik a keresést, ezekkel a szavakkal keresnek egyezéseket a különböző weboldalakon, dokumentumokban. Ezeknél a keresőknél a precision (pontosság), és a recall (fedés) értékei egyaránt alacsonyak. A precision azt mutatja meg százalékosan, hogy hány olyan találat volt, amely a felhasználó számára az adott keresés szempontjából értékes információkat tartalmazott. A recall azt mutatja meg százalékos formában, hogy mennyit sikerült megtalálni az összes olyan cikkből, dokumentumból, amelyek az adott keresés szempontjából relevánsak. A legnagyobb problémájuk, hogy rengeteg esetben nem képesek releváns találatokat biztosítani, mivel nem ismerik a keresett szavaknak a jelentéseit, ezáltal nem képesek kontextusba helyezni őket. Ez különösen a többjelentésű szavaknál, illetve a szinonímáknál okozhat jelentős problémát. A weboldalak sokszor bizonyos kulcsszavakat tudatosan azért helyeznek el az oldalon, hogy a kulcsszó alapú keresőmotorokkal egyszerűbben, gyorsabban megtalálható legyen az adott oldal. [10]

### 2.1.2 Szemantikus keresők

A szemantikus keresők ontológia segítségével garantálják, hogy a szavaknak a kapcsolatai és jelentései alapján releváns találatok kerüljenek megjelenítésre. A szemantikus keresők a szemantikus webet használják keresési térként, amely egy olyan jól definiált és jól strukturált információkat tartalmazó változata a webnek, ahol az egyes cikkek, weboldalak különböző szemantikus webes nyelvi elemekkel (RDF, OWL) egészülnek ki, amelyekkel kifejezhetőek a különböző kapcsolatok és összefüggések az egyes források és kulcsszavak között. A szemantikus keresők ezeknek az információknak a használatával képesek bonyolult keresésekre is releváns találatokat adni. [10]

Jelentős eltérés a kulcsszó alapú és a szemantikus keresők között az időtartam. A kulcsszó alapú keresők esetében az eredmények rendezése időigényes tud lenni a felhasználó számára, ezzel szemben a szemantikus keresők rövid idő alatt releváns találatokat tudnak adni, amelyek nem szorulnak rendezésre. [10]

### 2.1.3 NLP (Természetes nyelvfeldolgozás)

A természetes nyelvfeldolgozás, vagy röviden NLP egy olyan tudományterület, amely azt kutatja, hogy hogyan lehet a gépeket az emberi nyelven végzett kommunikáció, leírt szövegek értelmezésére felhasználni. Ezek a keresők az NLP technológia segítségével olyan találatokat képesek biztosítani, amelyek a keresett kulcsszavakat, kifejezéseket nem feltétlenül tartalmazzák, viszont ennek ellenére olyan adatokat tartalmaznak, amelyek kapcsolódhatnak azokhoz. A gépi tanulás ma már teljes mondatok elemzésére is használható, ezáltal a keresett kifejezésben lévő kulcsszavakat a mondat többi részét elemezve képes kontextusba helyezni. Ennek megvalósításához Open IE (Open Information Extraction) technológiát használnak, melynek lényege, hogy az alanyokat, állítmányokat, tárgyakat kinyerik a mondatból, így ezen információk ismeretében könnyebben értelmezhető, hogy mi történik az adott mondatban, mik az aktuális körülmények. [11]

Az NLP alapú keresők is küzdenek a többértelműség problámájával, ezeknél is megvan az esély arra, hogy a kereső félreértelmezi a felhasználó által megadott kifejezést, különösen a többjelentésű szavaknál. [12]

### 2.1.4 Faceted kereső

A faceted keresők, vagy más néven gyakorlatilag a szűrők koncepciója az, hogy a felhasználó különböző szempontok, állítások szerint kereshet egy adathalmazban. A felhasználó szemszögéből ez úgy néz ki, hogy különböző kategóriákat lát, amelyeket ki tud választani, és ez által ezek korlátfeltételekként jelennek meg a keresésben, tehát a találatok listája leszűkül azokra az egyedekre, amelyek az adott kategóriába sorolhatóak, rendelkeznek a kiválasztott tulajdonsággal. A szűrő egy-egy kérdése egy állítás (predikátum) szempontjából vizsgálja az egyedeket (pl. ha az adathalmaz egyedei emberek, akkor vizsgálhatóak nem vagy foglalkozás szempontjából). Ezekhez a szempontokhoz több féle válaszlehetőség tartozhat, tehét az egyedek több féle értéket vehetnek fel az adott szempontok szerint vizsgálva, ezért a szűrő működéséhez az adathalmaznak predikátum-értékpárokat kell tartalmaznia. [13]

A kategóriák kialakításának a módja történhet hagyományos módon, vagy ha szemantikával ötvözik, akkor ontolódia segítségével is. A kategóriák kialakításával kapcsolatban egy olyan probléma merülhet fel, hogy a felhasználók lehetséges, hogy máshogy alakítanák ki őket, hiszen számukra nem túl természetes az aktuális felosztás. Ez lehet amiatt, mert túl általánosan kerültek kialakításra, vagy akár pont az ellenkezője miatt is, mikor túl szakmaiak lettek a kategóriák, amelyeket pedig a felhasználó már nem ért. [14]

### 2.1.5 Kép alapú keresők (CBIR)

A kép alapú keresők kevésbé sorolhatók a hagyományos keresők közé. A kép alapú keresők a képeket azoknak a vizuális jellemzői, például szín, alak, textúra alapján keresik vissza. Régebben a képindexelés úgy működött, hogy a képekhez különböző kulcsszavakat, leírásokat rendeltek hozzá a visszakereshetőség érdekében, viszont ez rendkívül lassú, munka és időigényes volt. Erre a problémára jelentettek megoldást a kép alapú keresők, ahol ilyen jellegű munkálatokra már nem volt szükség ahhoz, hogy kereshetőek legyenek a képek. [15]

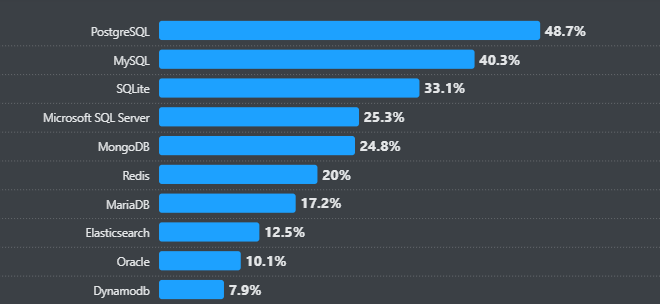
A CBIR folyamata során a képek keresése a következő képpen néz ki: Először megadja a felhasználó azt a képet, amelyhez hasonló képeket szeretne keresni. A kereső elemzi ezt a képet a vizuális tulajdonságai alapján, majd az összes képével ugyanezt teszi annak az adatbázisnak, amelyben keresünk. Ezután a keseséshez használt kép jellemzőit összeveti az adatbázis képeinek jellemzőivel, és a találatok között azok a képek fognak szerepelni, ahol vizuálisan hasonló eredményt kapunk. [15]

## 2.2 SQLite bemutatása

Az SQLite [7] egy olyan nyílt forráskódú C nyelvű könyvtár, amely egy cross-platform, teljes körű SQL adatbázis-motort valósít meg. Jellemzői közé tartozik, hogy rendkívül kicsi, gyors, önálló, és nagy megbízhatóságú. [16]

Az SQLite a világ egyik leggyakrabban használt adatbázis-motorja. [1. ábra] A cross-platform tulajdonságának köszönhetően számos különböző eszközön, köztük mobiltelefonokban, televíziókban, játékkonzolokban, kamerákban, autókban, és a legtöbb számítógépben is megtalálható valamilyen formában, vagy beépítve, vagy pedig olyan alkalmazások részeként, amelyeket az emberek nap mint nap használnak ezeken az eszközökön. [17]

A StackOverflow 2024-es felmérése [18] alapján a válaszokból ítélve a felhasználók 33,1%-a válaszolta azt, hogy használja az SQLite-ot, ezzel a listán a 3. helyet szerezte meg a legnépszerűbb, legelterjedtebb adatbázisok között. [1. ábra]



**1.** ábra: A StackOverflow 2024-es felmérése különböző adatbázis motorok elterjedtségéről 3

A listában csak a PostgreSQL és a MySQL előzik meg, viszont ezek nehezen tekinthetőek az SQLite konkurenseinek, mivel ezek kliens/szerver adatbázis motorok, amelyek más jellegű problémákra nyújtanak megoldást. Általánosságban elmondható, hogy ezeknél az elosztott vállalati adatoknak egy központosított adatbázisban való tárolása a cél. Ezzel szemben az SQLite arra a célra készült, hogy helyi tárolás céljából biztosítson egy kisebb méretű adatbázist különböző alkalmazásokhoz, programokhoz, amelyben egyszerűen és gyorsan tárolhatók azok az adatok, amelyekkel a program dolgozik. [17]

Mivel a szakdolgozatom középpontjában egy helyi, C# program áll, így a kliens/szerver adatbázis motorok helyett egy kisebb méretű helyi adatbázisra volt szükségem, ezért is esett a választásom az SQLite-ra.

Ettől eltekintve az SQLite nem csak helyi programok esetében használható, sőt kliens/szerver adatbázismotort igénylő feladatok is megvalósíthatóak vele, vagy akár egy közepes látogatottságú weboldal adatbázisaként is használható. [17]

Mivel egy C# nyelvű programról van szó, ezért az adatbázismotor kiválasztásánál egy fontos kitétel volt, hogy C# nyelvvel könnyen, és jól használható legyen. A *System.Data.SQLite* egy ADO.NET szolgáltatás az SQLite-hoz. [19] Az SQLite adatbázismotor ennek a *System.Data.SQLite* NuGet csomagnak a segítségével könnyedén használható, és integrálható bármely C# alkalmazásba, így tökéletes választásnak bizonyult egy SQLite adatbázis a szakdolgozatom programjához szükséges adatok eltárolásához.

## 2.3 Az adatbázis felépítése

Az adatbázis 11 darab táblát tartalmaz, amelyek a következőek: *Movies, Countries, Directors, Genres, Keywords, Languages, Movies\_Countries, Movies\_Directors, Movies\_Genres, Movies\_Keywords, Movies\_Languages.*

### 2.3.1 Az adatbázis szerkezete

A **Movies** tábla a következő mezőkből áll:

* ID:
  + Típus: INTEGER
  + Leírás: A film egyedi azonosítója, ez a primary key (elsődleges kulcs).
* Title:
  + Típus: TEXT
  + Leírás: A film címe angolul, amely az esetek nagy részében megegyezik a filmek eredeti címével.
* Genre:
  + Típus: TEXT
  + Leírás: A film műfaja(i) felsorolásszerűen, vesszővel elválasztva. Egyes filmeknél csak 1 darab van belőle, más, több műfajba is beleillő filmeknél több is lehet.
* Released:
  + Típus: INTEGER
  + Leírás: A film megjelenésének éve.
* Runtime:
  + Típus: INTEGER
  + Leírás: A film játékideje percben megadva.
* Gender\_of\_the\_protagonist:
  + Típus: INTEGER
  + Leírás: A főszereplő nemét jelzi. Az értéke 1, ha a főszereplő nő, ha férfi akkor pedig 2 az értéke.
* Main\_actor:
  + Típus: TEXT
  + Leírás: A főszereplő színész neve.
* Keywords:
  + Típus: TEXT
  + Leírás: Az adott filmhez kapcsolódó, a TMDB-ről származó kulcsszavak vesszővel ellátott felsorolása, melyek használatával könnyebben találhatunk hasonlóságokat a filmek között.
* Director:
  + Típus: TEXT
  + Leírás: Az adott film rendezőjének/rendezőinek neve.
* Language:
  + Típus: TEXT
  + Leírás: A film eredeti nyelve, illetve a benne elhangzó nyelvek felsorolása.
* Pruduction\_countries:
  + Típus: TEXT
  + Leírás: A készítő országok neveinek felsorolása.
* Tmdb\_score:
  + Típus: REAL
  + Leírás: Az adott film TMDB-n szereplő értékelése, amely a felhasználói értékelések átlaga. Az értékelés egy 0-tól 10-ig terjedő skálán történik, így a pontszám is a két érték közé eshet csak.
* Number\_of\_ratings:
  + Típus: INTEGER
  + Leírás: Megmutatja, hogy a TMDB oldalán eddig hányan értékelték az adott filmet, tehát a „Tmdb\_score” nevű mező hány értékelésnek az átlaga.
* Popularity:
  + Típus: REAL
  + Leírás: Az adott film TMDB szerinti népszerűségi indexe, melyet a felhasználók tevékenységeiből számolnak.
* Budget:
  + Típus: INTEGER
  + Leírás: Az adott film költségvetésének összege amerikai dollárban megadva.
* Revenue:
  + Típus: INTEGER
  + Leírás: Az adott film bevételének összege amerikai dollárban megadva.

A **Countries** tábla a következő mezőkből áll:

* ID:
  + Típus: INTEGER
  + Leírás: Az adott ország egyedi azonosítója, ez az elsődleges kulcs.
* Country\_Name:
  + Típus: TEXT
  + Leírás: Az adott ország neve.

A **Directors** tábla a következő mezőkből áll:

* ID:
  + Típus: INTEGER
  + Leírás: Az adott rendező egyedi azonosítója, ez az elsődleges kulcs.
* Director\_Name:
  + Típus: TEXT
  + Leírás: Az adott rendező neve.

A **Genres** tábla a következő mezőkből áll:

* ID:
  + Típus: INTEGER
  + Leírás: Az adott műfaj egyedi azonosítója, ez az elsődleges kulcs.
* Genre\_Name:
  + Típus: TEXT
  + Leírás: Az adott műfaj neve.

A **Keywords** tábla a következő mezőkből áll:

* ID:
  + Típus: INTEGER
  + Leírás: Az adott kulcsszó egyedi azonosítója, ez az elsődleges kulcs.
* Keyword\_Name:
  + Típus: TEXT
  + Leírás: Az adott kulcsszó neve.

A **Languages** tábla a következő mezőkből áll:

* ID:
  + Típus: INTEGER
  + Leírás: Az adott nyelv egyedi azonosítója, ez az elsődleges kulcs.
* Language\_Name:
  + Típus: TEXT
  + Leírás: Az adott nyelv elnevezése.

A **Movies\_Countries** kapcsolótábla a következő mezőkből áll:

* Movie\_ID:
  + Típus: INTEGER
  + Leírás: Az adott film egyedi azonosítója, idegenkulcs, amely a *Movies* tábla ID mezőjére mutat.
* Countries\_ID:
  + Típus: INTEGER
  + Leírás: Az adott ország egyedi azonosítója, idegenkulcs, amely a *Country* tábla ID mezőjére mutat.

A **Movies\_Directors** kapcsolótábla a következő mezőkből áll:

* Movie\_ID:
  + Típus: INTEGER
  + Leírás: Az adott film egyedi azonosítója, idegenkulcs, amely a *Movies* tábla ID mezőjére mutat.
* Directors\_ID:
  + Típus: INTEGER
  + Leírás: Az adott rendező egyedi azonosítója, idegenkulcs, amely a *Director* tábla ID mezőjére mutat.

A **Movies\_Genres** kapcsolótábla a következő mezőkből áll:

* Movie\_ID:
  + Típus: INTEGER
  + Leírás: Az adott film egyedi azonosítója, idegenkulcs, amely a *Movies* tábla ID mezőjére mutat.
* Genres\_ID:
  + Típus: INTEGER
  + Leírás: Az adott műfaj egyedi azonosítója, foreign key, amely a *Genre* tábla ID mezőjére mutat.

A **Movies\_Keywords** kapcsolótábla a következő mezőkből áll:

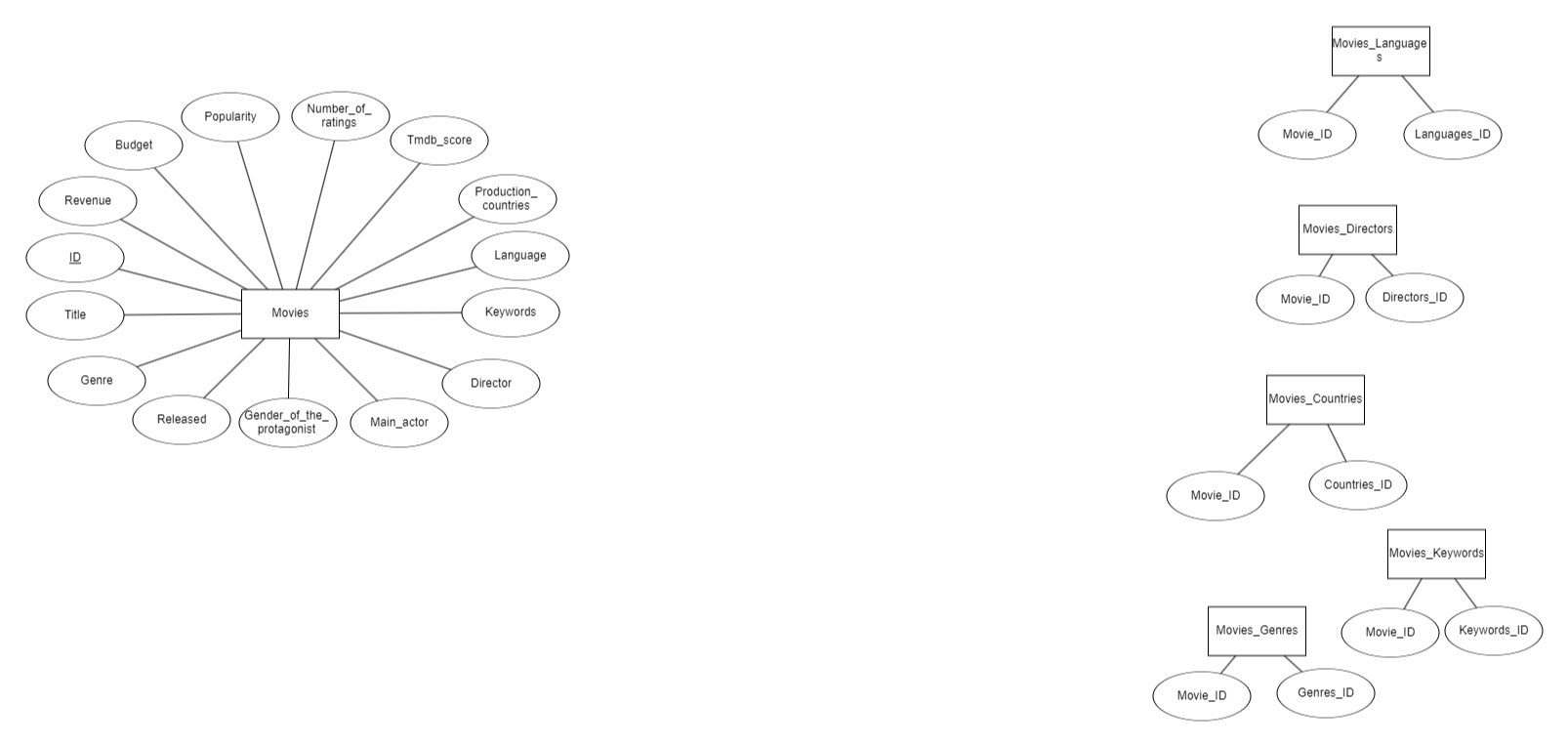
* Movie\_ID:
  + Típus: INTEGER
  + Leírás: Az adott film egyedi azonosítója, idegenkulcs, amely a *Movies* tábla ID mezőjére mutat.
* Keywords\_ID:
  + Típus: INTEGER
  + Leírás: Az adott kulcsszó egyedi azonosítója, idegenkulcs, amely a *Keyword* tábla ID mezőjére mutat.

A **Movies\_Languages** kapcsolótábla a következő mezőkből áll:

* Movie\_ID:
  + Típus: INTEGER
  + Leírás: Az adott film egyedi azonosítója, idegenkulcs, amely a *Movies* tábla ID mezőjére mutat.
* Languages\_ID:
  + Típus: INTEGER
  + Leírás: Az adott nyelv egyedi azonosítója, idegenkulcs, amely a *Languages* tábla ID mezőjére mutat.

### 2.3.2 Az adatbázis ER modellje

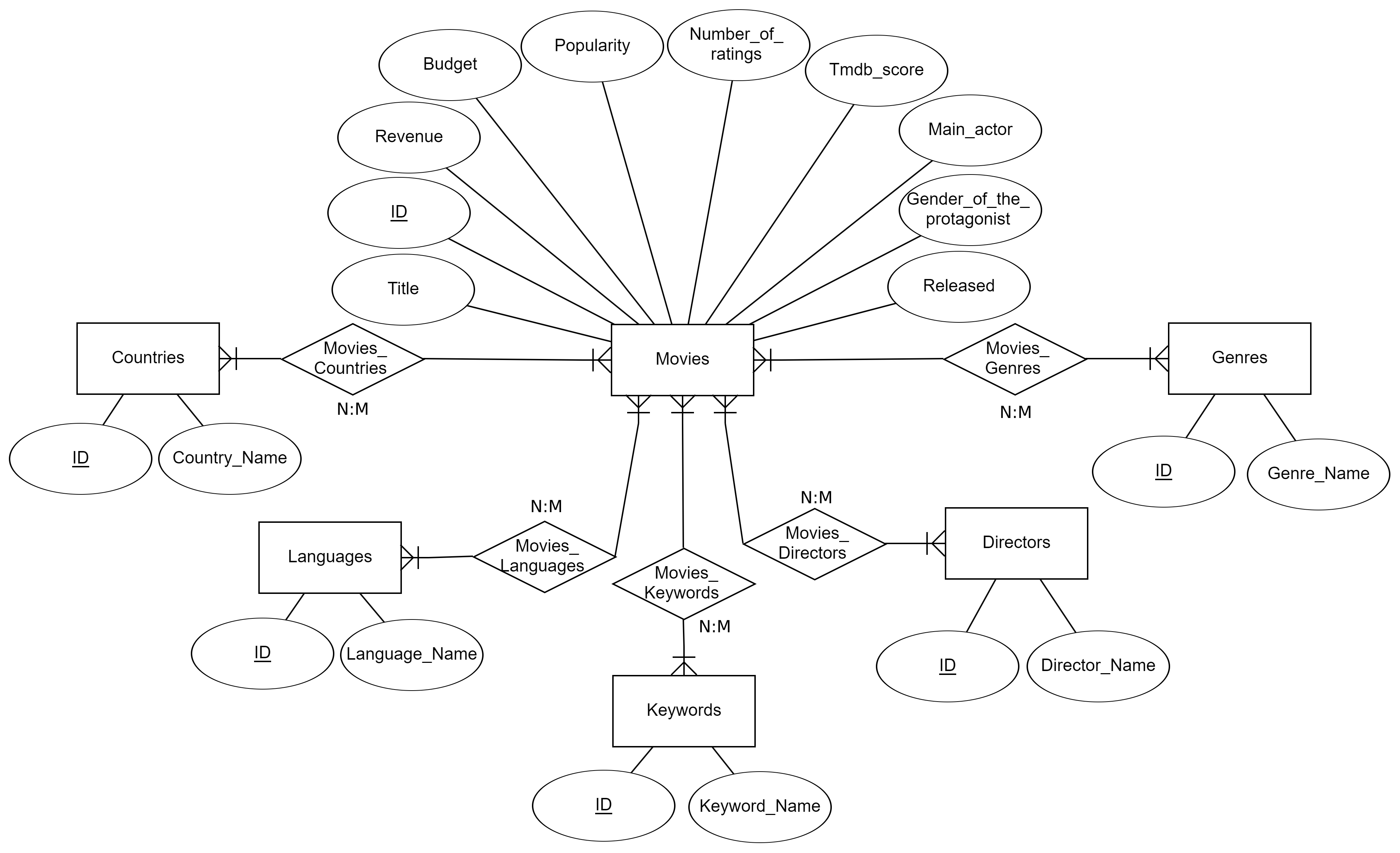
Az adatbázisok szerkezeti felépítésének szemléltetésére két gyakran használt módszer van, ezek közül az egyik az ER modell. A diagramon a táblákat téglalapok jelölik, benne a nevükkel, a táblák mezőit pedig ellipszisek, amelyek belsejében szintén megtalálható a nevük is. A mezők vonalakkal vannak összekötve azzal a táblával, amely tartalmazza őket. Az adatbázis kiindulási ER modelljén [2. ábra] jól látszik, hogy az adatbázis csupán egyetlen táblát tartalmaz, és az összes mező ehhez a táblához kapcsolódik. Az ID mező neve aláhúzással van megjelőlve, ez azt szemlélteti, hogy a táblának ez az elsődleges kulcsa.

****

**2.** ábra: A movies.db adatbázis kiindulási, lebontás előtti ER modellje

A *Movies* tábla adattagjainak a tartalmát vizsgálva a legtöbb egy egyszerű értéket tartalmaz, azonban vannak közöttük olyan mezők is, amelyek több értéket is tartalmaznak, egymástól vesszővel elválasztva. Ezeket az értékeket ebben a formában nem lehet hatékonyan felhasználni, mivel a mezők különböző értékeit nem tudjuk külön-külön vizsgálni, csak egyben. Pl. ha egy filmben 3 nyelven is megszólalnak, ezáltal a nyelve „Angol, Magyar, Német”, akkor ebben a formában erről a filmről csak ennyit tudunk elmondani. Viszont ha külön-külön arról szeretnénk információt kapni, hogy az adott filmben beszélnek-e „Angol” vagy „Magyar” nyelven, akkor az adatbázisban nincs olyan információ, ami konkrétan erre a kérdésre adna választ, ez pedig megnehezíti az adatok, jelen esetben filmek ezen tulajdonságokon alapuló egymással történő összevetését.

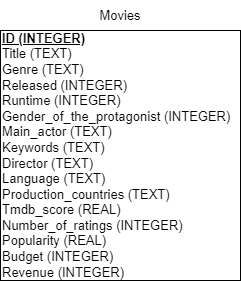
Az érintett mezők tartalmait lebontva az adatbázis szerkezete nagyban megváltozik, melyet a lebontás utáni ER modell [3. ábra] jól szemléltet. Az érintett mezők már nem csak közvetlenül a Movies táblához tartoznak (az adatbázis szerkezetében továbbra is jelen vannak ott is, de az ábra bonyolultságának elkerülése érdekében nincsenek ábrázolva), hanem külön táblákká alakultak át, és ezeket a közöttük lévő N:M típusú kapcsolatok kapcsolják a Movies táblához. Mivel a kapcsolatok mintkét irányba több-több (N:M), így ennek tényleges megvalósításához kapcsolótáblák létrehozása szükséges, viszont ez nem látszik az ER diagramon.



**3**. ábra: A movies.db adatbázis lebontás utáni ER modellje

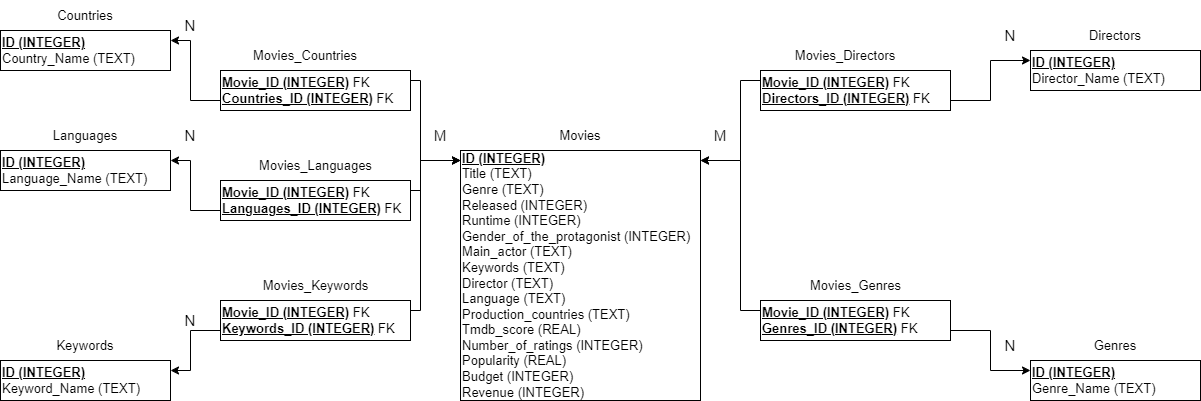
### 2.3.3 Az adatbázis relációs modellje

Az adatbázisok szerkezeti felépítésének másik gyakori módszere a relációs modell. Az adatbázis szerkezetének az állapotát az előbbiekben részletezett lebontás folyamatát megelőzően a kiindulási relációs modell [4. ábra] mutatja meg. A relációs modellben is téglalapok jelzik a táblákat, viszont itt a mezők a téglalapba beleírva helyezkednek el. A kiindulási relációs adatmodell annyival tartalmaz több információt az adatbázisról a kiindulási ER modellel [2. ábra] szemben, hogy itt az is jelölve van, hogy a különböző mezők milyen típusú adatokat tartalmaznak.



**4.** ábra: A movies.db adatbázis kiindulási, lebontás előtti relációs modellje

Az adatbázis szerkezetének a lebontás folyamata utáni állapotát szemléltető relációs modellje [5. ábra] már részletesebb képet ad az adatbázisról, hiszen itt már megjelennek a kapcsolótáblák, és azok mezői is, továbbá szintén megjelennek a mezőkben tárolt adatok típusai is.



**5**. ábra: A movies.db adatbázis lebontás utáni relációs modellje

A lebontás során 1 táblából 11 táblánk lesz. Minden lebontott mezőhöz 1 tábla és egy kapcsolótábla is fog tartozni.

Minden lebontott tábla N:M kapcsolatot valósít meg a *Movies* táblával, hiszen pl. egy filmnek lehet több műfaja, illetve ha van egy műfaj, akkor az több filmnél is szerepelhet, nem csak egy darabnál. Egy filmben beszélhetnek több nyelven is, továbbá több filmben is beszélhetnek ugyanolyan nyelven, így ugyanazt a nyelvet több filmhez is hozzárendelhetjük, és így tovább. Az ilyen jellegű több-több kapcsolat miatt van tehát szükség a kapcsolótáblákra minden lebontott mező esetében.

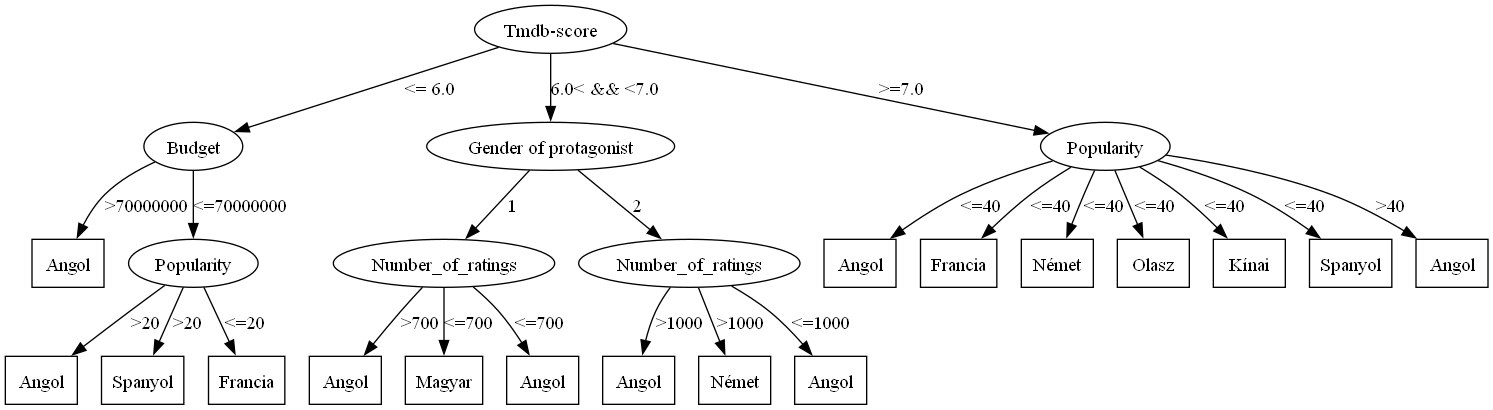
A táblák közötti kapcsolatok mindkét irányba kötelezőek: Minden filmnek van műfaja, gyártási országa, rendezője, stb., ezért ez kötelező jellegű kapcsolat. A lebontott tulajdonságok esetében, tehát pl. a műfajokat, rendezőket, kulcsszavakat, stb. tartalmazó táblákban alapvetően előfordulhatnának olyan adatok, amelyekhez nem rendelünk filmet, hiszen egyik film sem rendelkezik ilyen tulajdonságokkal. Viszont a program felépítéséből adódóan ez nem lehetséges, hiszen ezeket a táblákat az adatbázisban lévő filmek adatait használva töltjük fel, és amennyiben ezek közül bármelyik film, vagy annak adata törlődik, akkor az összes ilyen jellegű tábla tartalmát frissítjük. Tehát pl. a *Genres* táblát nézve annak minden egyes műfajához fog legalább egy film kapcsolódni a *Movies* téblában.

## 2.4 Kézzel készített döntési fa a film nyelvére vonatkozóan

A szakdolgozatom témája a kérdőív alapú adaptív ajánlórendszerek, melyekhez szorosan kapcsolódik a döntési fák témaköre is. Az ajánlórendszerek olyan gépi tanulási technikák, amelyek különböző termékek, szolgáltatások sokaságát tudják javasolni a felhasználó számára a felhasználótól származó információk alapján, ezáltal könnyebbé teszi számára az ő ízlésének megfelelő termékek, szolgáltatások megtalálását. [3]

A beérkező információk feldolgozásának egyik módszere a döntési fák, melyek nagy pontossággal képesek osztályozni a különböző információkat, így tökéletes alapot tudnak szolgáltatni az ajánlórendszer döntéseinek meghozatalában. [4]

A döntési fák szemléltetésére létrehoztam egy olyan fát [6. ábra], amely kézzel került összerakásra, a leírásához JSON formátumot választottam. Azért a JSON formátumra esett a választásom, mert egy olyan platformfüggetlen formátumról van szó, amelyet szinte minden programozási nyelv támogat. Egyszerű a szintaktikája, a használatával leírt adatok, modellek egyszerűen feldolgozhatóak más programozási nyelvek segítségével, továbbá érthetőek az emberek és gépek számára is egyaránt. [20]



**6.** ábra: A film nyelvére vonatkozó kézzel készített fa ábrázolása

A fa egy 20 elemű tanító halmaz segítségével került kialakításra, ami az adatbázisom utolsó 20 db filmjéből áll, azok alapján alakítottam ki az elágazásokat. A JSON fájl felépítésében 3 ismétlődő elem jelenik meg: a „node”, amely a csomópontokat jelöli, a „condition”, ami a feltételeket, elágazásokat jelöli, illetve az „outcome”, amelyek a fa levelei, tehát a lehetséges kimenetek.

Az alábbi kódrészlet a döntési fa egyik ága, amelyet akkor vizsgálunk, ha az adott film TMDB értékelése 6,0, vagy annál kisebb. Ezután a következő csomópont a *Budget*: ha nagyobb, mint 70000000, akkor a film nyelve Angol. Ha kisebb, vagy egyenlő, mint 70000000, akkor még a film *Popularity* tulajdonságának vizsgálatára is szükség van. Ha ez az érték nagyobb, mint 20, akkor a film nyelve, tehát a fa kimenete Angol és Spanyol, ellenben ha kisebb, vagy egyenlő mint 20, akkor Francia.

{

"decision\_tree": {

"root": {

"node": "Tmdb-score",

"condition": [

{

"condition": "<= 6.0",

"outcome": {

"node": "Budget",

"condition": [

{

"condition": ">70000000",

"outcome": "Angol"

},

{

"condition": "<=70000000",

"outcome": {

"node": "Popularity",

"condition": [

{

"condition": ">20",

"outcome": ["Angol", "Spanyol"]

},

{

"condition": "<=20",

"outcome": "Francia"

}

]

}

}

]

}

}

## 2.5 Kézzel készített döntési fa hatékonysága

A bemutatott döntési fa hatékonyságának mérését az 1. táblázat mutatja be.

**1.** táblázat: A kézzel készített döntési fa hatékonyságának mérési eredményei

|  |  |  |  |
| --- | --- | --- | --- |
| Intervallum | "1-20" | "60-80" | Összesen |
| Helyes | 17 | 4 | 21 |
| Összesen | 20 | 20 | 40 |
| Hatékonyság | 85,00% | 20,00% | 52,50% |

A fa hatékonyságának mérését a *movies.db* adatbázis két különböző, de egyforma méretű tartományán végeztem. A mérésnél az egyes eseteknél abban az esetben soroltam „Helyes” kategóriába az adott esetet, ha a film nyelveit a különbőző tulajdonságainak vizsgálatával, a döntési fát használva pontosan sikerült megállapítani. A többnyelvű filmeknél, pl. a "Magyar, Angol"-t egy nyelvnek vettem, hiszen a döntési fa kialakításánál is ahol több nyelv volt jelen egy filmben, azokat együttesen vettem egy nyelvként. Tehát pl. ha egy film nyelve „Angol, Spanyol”, de a döntési fát használva végeredményként amennyiben csak az „Angol”, vagy esetleg csak a „Spanyol” eredmény jött ki végeredménynek, akkor az adott filmre vonatkozó becslés nem bizonyult „Helyes” eredményűnek.

Az 1. táblázat eredményeiből jól látszik, hogy a hatékonyság nagyban függ attól, hogy az adatbázis melyik tartományában vizsgáljuk a filmeket. Hatékonyságra az első 20 db filmet vizsgálva 85% jött ki eredményként, mivel az adatbázis ezen szakaszán nagyrészt angol nyelvű filmek voltak csak, így itt jobban működött a fa. A 61-80. filmre is megnéztem a hatékonyságát, itt láthatóan kevésbé működött jól a fa, csak 20%-os hatékonyságot adott, ami azzal magyarázható, hogy ebben az intervallumban sokkal vegyesebb nyelvű filmek szerepelnek, illetve sok olyan nyelvkombináció (pl. "Afrikai, Angol", "Angol, Japán") jelent meg, amelyek a döntési fában nem szerepeltek, hiszen az utolsó 20 filmnél nem voltak ezek jelen. A két szeletet együttvéve, a 40 filmet összességében vizsgálva 52,5%-os hatékonyság jön ki eredményként.

Később a programom *DecTreeForLanguage* osztályában létrehoztam egy *CompareToHandmadeDecTree* metódust, amelyben az ML.NET segítségével létrehoztam egy hasonló becslést, mint a kézzel készített döntési fa. A metódus célja, mint a neve is jelöli, az volt, hogy össze lehessen hasonlítani egy gép által létrehozott becslést a kézzel készített döntési fával. Ezt úgy valósítottam meg, hogy a filmek ugyanazon jellemzőit adtam meg betanítási adatnak, mint amik a kézzel készített fában is szerepelnek, tehát a *Budget, Popularity, NumberOfRatings, GenderOfProtagonist* és a *TmdbScore* jellemzőket. A kimenet, tehát a jellemző, amire a becslés irányult a film nyelve volt értelemszerűen itt is. Betanításhoz ugyanúgy az adathalmaz utolsó 20 filmjét használtam fel, mivel ezek alapján készült ugye a kézzel készített fa is. A hatékonyság vizsgálatakor ugyanúgy 2 tartományt vizsgáltam: az első 20 filmet, illetve a 60. és 80. film közti intervallumban lévő filmeket. A kapott eredményeket a 2. táblázat tartalmazza.

**1.** táblázat: Az ML.NET-tel készített becslő algoritmus hatékonyságának mérési adatai

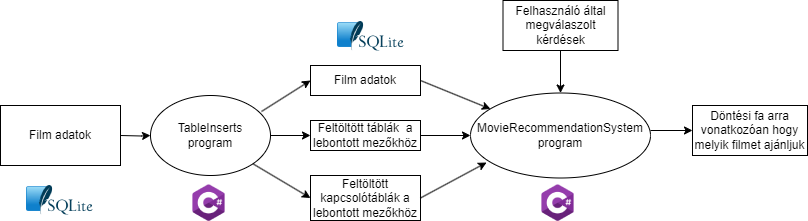
|  |  |  |  |
| --- | --- | --- | --- |
| Intervallum | "1-20" | "60-80" | Összesen |
| Helyes | 17 | 7 | 24 |
| Összesen | 20 | 20 | 40 |
| Hatékonyság | 85,00% | 35,00% | 60,00% |

A mérés módja és szempontjai teljes mértékben megegyeztek a korábban, a fa mérésénél ismertetett módszerrel, csak jelen esetben automatizálva történtek. A *CompareToHandmadeDecTree* metódusban a becsült értékeket összevetettem a valós értékekkel, és ha egyezést találtam, akkor egy erre a célra létrehozott double típusú változó értékét megnöveltem. Miután végigment a program a filmeken, a kapott értéket elosztottam 20-szal, mivel egy 20 elemű intervallumot vizsgáltunk, így megkaptam a hatékonyságot százalékos formában.

A mérési adatok összehasonlításából az látszik, hogy az első vizsgált intervallumban ugyanolyan hatékonysággal tudott működni az algoritmus, mint a kézzel készített fa, tehát 85%-os hatékonysággal. A második intervallumban, amely vegyesebb nyelvű filmeket tartalmaz, viszont tudott javulni a kézzel készített becsléshez képest, mivel a 20 filmből 4 helyett 7 darabot sikerült helyesen megbecsülnie, ezzel pedig a teljes hatékonyságot 52,5%-ról 60%-ra sikerült növelni. Tehát ezek alapján a géppel készített becslő algoritmus 7,5%-kal hatékonyabb, mint a kézzel készített döntési fa.

## 2.6 Folyamatábra az adatbázis adatainak feldolgozásáról

Az 7. ábra szemlélteti azt, hogy hogyan, és milyen eszközökkel jutunk el az adatázisba összegyűjtött filmektől a felhasználó számára ajánlott filmekig. A folyamatban a feldolgozóegységek a C# nyelven írt programok. Azon belül a TableInserts egy egyszerű konzolos, tehát grafikus felülettel nem rendelkező program, amely a korábbiakban részletezett mezők lebontásáért felelős, tehát gyakorlatilag egy segédprogramként szolgál a fő program számára, hiszen a feldolgozáshoz szükséges formába hozza a filmek adatait. A lebontással új táblák jelennek meg az eredeti *Movies* tábla mellett, ezekkel egészül ki az eredeti, SQLite formátumban tárolt adatbázis. A fő grafikus program, amely az adatok feldolgozását, kezelését, megjelenítését, illetve magát az ajánlórendszert is megvalósítja, már a TableInserts által átalakított, előkészített adatbázist kapja meg bemenetként. Másik bemeneti forrása a felhasználó által adott információk, a megválaszolt kérdések, hiszen ezek felhasználásával, feldolgozásával fogja elvégezni a program az ajánlott filmekre vonatkozó becslését, amely majd a program kimenetéül fog szolgálni.

****

**7**. ábra: Folyamatábra a feldolgozandó adatokról, és a feldolgozóegységekről

## 2.7 ML.NET bemutatása

### 2.7.1 A mesterséges intelligencia és a gépi tanulás kapcsolata

A mesterséges intelligencia egy olyan ága a számítástechnikának, amely azzal foglalkozik, hogy a számítógépeket olyan feladatok elvégzésére „tanítja be”, amelyekhez általában valamilyen emberi beavatkozás, emberi intelligencia szükséges. Napjainkban megkérdőjelezhetetlen ütemben fejlődik és terjed, az élet egyre több területén jelenik meg a használata, és folyamatosan beépítésre kerül a különböző szolgáltatásokba, rendszerekbe. A mesterséges intelligenciának több ágazata is van, ezek közül egyik meghatározó ága a gépi tanulás. [21]

A gépi tanulás alatt egy olyan folyamatot értünk, amely automatikusan képessé tesz egy gépet vagy rendszert arra, hogy tanuljon és javuljon a feldolgozott adatokból vett tapasztalatokból, ezáltal előrejelzéseket, becsléseket tud készíteni az új adatokra vonatkozóan. Tehát az explicit programozás helyett a gépi tanulás algoritmusokat használ nagy mennyiségű adat elemzésére, a felismerésekből való tanulásra, majd ezek alapján a döntések meghozatalára. [21]

A gépi tanulási algoritmusok teljesítménye idővel javulni tud azáltal, hogy egyre több adattal találkoznak amelyeket elemeznek, majd mintákat ismernek fel belőle, amiknek a megtanulásával egyre pontosabb eredményeket tudnak elérni. [21]

Maga a gépi tanulási modell alatt azt értjük, amit a program az algoritmus a tanító adatokon történő futtatásából tanul. Minél több adatot használunk, annál jobb, pontosabb lesz a modell. [21]

A térhódításnak megfelelően a különböző programozási nyelvekhez is megjelentek az utóbbi években a különböző kiegészítők, függvénykönyvtárak, amelyek lehetővé teszik a már előre implementált gépi tanulást használó programrészek, algoritmusok beépítését a programjainkba.

A legelterjedtebb programozási nyelvek közül párat említve a C++-hoz az mlpack [22], a pythonhoz a scikit-learn [23] függvénykönyvtárak használhatóak a gépi tanulás algoritmusainak eléréséhez, az általam is használt C# nyelv esetén pedig az ML.NET [24] függvénykönyvtár tartalmazza ezeket az algoritmusokat.

### 2.7.2 Mi az az ML.NET?

Az ML.NET [24] egy ingyenes, nyílt forráskódú és cross-platform, azaz Windows, Linux és macOS operációs rendszerekre is egyaránt elérhető gépi tanulási keretrendszer a .NET fejlesztői platformhoz. A keretrendszer nem csak C#, hanem a kevésbé ismert F# nyelvvel is kompatibilis. [25] Az ML.NET segítségével betanítható egy egyéni modell, vagy akár importálhatóak előre betanított TensorFlow- és ONNX-modellek. [26]

Az ML.NET tökéletesen használható számos kategorizálási, előrejelzési, összehasonlítási feladatra, többek között például az ügyfelek visszajelzéseinek kategorizálására (jó vagy rossz), folytonos jellemzők, tehát például termékek árának megbecslésére, dolgok hasonlóságának összehasonlítására, vagy akár a felhasználók korábbi tevékenységei, érdeklődési körei alapján történő ajánlások készítésére, ami az én szakdolgozatomnak is a feladata. [26]

A függvénykönyvtár első verziójának kiadására 2018 május 7-én került sor, a stabil kiadás legelső, 1.0-ás verzióját pedig 1 évvel később, 2019-ben a Microsoft Build nevű konferenciáján jelentették be. [25] A szakdolgozatom írásakor a legfrissebb kiadott stabil verzió az a 2024 január 18-án megjelent 3.0.1-es verzió, így a program megírásához is ezt a verziót használtam.

## 2.8 Hogyan néz ki egy ML.NET alkalmazás?

Egy ML.NET program felépítése általánosságban a következőképpen néz ki:

### 2.8.1 MLContext létrehozása

Egy ML.NET alkalmazás legelső lépése az MLContext létrehozása, hiszen ezek után kezdődhet csak el az adatok betöltése, átalakítása, a modell betanítása, az algoritmusok kiválasztása. Tehát az MLContext az egész ML.NET alkalmazás kiindulópontja. [27]

var mlContext = new MLContext();

(Az ML.NET dokumentációjából [27] származó kódrészletek)

### 2.8.2 Adatok betöltése

Mint már korábban is említésre került, a gépi tanulás már ismert adatokat dolgoz fel, ezekben keres mintákat, majd az ezekből nyert ismeretek segítségével becsül meg ismeretlen adatokat, információkat. Tehát az ML.NET-nek is szüksége van ezekre az úgynevezett betanító adatokra, amiket fel tud használni a későbbi becslések során.

A bemeneti adatokat, amikben a minták keresése, felismerése történik, Feature-öknek nevezzük, így jelennek meg a kódban. A kimeneti adatok, amelyeknek az előrejelzése gyakorlatilag a programnak a célja, Label-ként szerepelnek a kódban. [27]

Az adatok IDataView objektumokba kerülnek betöltésre, amelyek tartalmazhatnak különböző típusú számokat, szövegeket, logikai értékeket, vagy akár vektorokat és számokat tartalmazó tömbőket is tölthetünk bele, listákat viszont nem támogat. Az adatok betöltése, amennyiben fájlból történik, lehetséges többek között txt, CSV, TSV és egyéb formátumokból is, viszont élő, valós idejű adatok betöltésére is lehetőség van. [27]

IDataView trainingData = mlContext.Data.LoadFromTextFile<SentimentInput>(dataPath, separatorChar: ',', hasHeader: true);

Amennyiben már memóriában lévő, változókból, tömbökből származó adatokat szeretnénk betölteni, akkor arra is lehetőségünk van. [27]

IDataView trainingData = mlContext.Data.LoadFromEnumerable<SentimentInput>(inMemoryCollection);

### 2.8.3 Adatok átalakítása

Sok esetben az adatok hiába tartalmaznak rengeteg hasznos és értékes információt, eredeti formájukban, amikben betöltésre kerültek sajnos nem használhatóak, így átalakításokat kell végeznünk rajtuk annak érdekében, hogy a modellünk betanítására fel tudjuk őket használni. [27] A szövegként tárolt értékek egy részét például számokká kell átalakítanunk, a különböző hosszúságú tömböket pedig ahhoz, hogy összehasonlíthatóak legyenek, elő kell készítenünk olyan módon, hogy egységes elemszámúak legyenek.

Számos lehetőséget kínál a függvénykönyvtár az adatok átalakítására. Az átalakításokat végző metódusok egy részének feltétele az, hogy csak a betanítási adatokra lehet őket meghívni, egyes adatátalakításokhoz viszont nem szükségesek a betanítási adatok, így használhatóak azok nélkül is. [28] A teljesség igénye nélkül néhány, amelyek gyakoribb problémákat oldanak meg:

* ConvertType: A bemeneti oszlop típusát lehet vele átalakítani.
* MapValueToKey: A művelet során a bemeneti adatok értékeit egy adott kategóriákhoz (kulcsokhoz) rendeljük, tehát minden kategória egyedi kulcsértékkel fog rendelkezni, amelyek alapján azonosítani tudjuk majd őket.
* TokenIntoWords: Egy vagy több darab, szöveget tartalmazó oszlop osztható fel vele szavakra.
* OneHotEncoding: Segítségével a kategóriákat különálló, bináris vektorokba kódoljuk, ami lehetővé teszi a szöveges adatok hatékony feldolgozását és elemzését.
* Concatenate: Segítségével megoldhatjuk azt a problémát, hogy ha több oszlop adatait egyetlen oszlopba szeretnénk egyesíteni, például szöveges adatok esetén.

Az alábbi kódrészlet az adatok átalakítására mutat egy példát:

var dataProcessPipeline = mlContext.Transforms.Conversion.ConvertType(nameof(TMDB.TMDBScore), nameof(TMDB.TMDBScore), DataKind.Single)

.Append(mlContext.Transforms.Conversion.ConvertType("KeywordFloat", nameof(TMDB.Keyword), DataKind.Single))

.Append(mlContext.Transforms.Conversion.ConvertType("GenreFloat", nameof(TMDB.Genre), DataKind.Single))

.Append(mlContext.Transforms.Concatenate("Features", "KeywordFloat", "GenreFloat"))

.AppendCacheCheckpoint(mlContext);

Ebben az esetben a becsléshez 2 bemeneti adattag kerül felhasználásra, a *Keyword* és a *Genre*, ezek alapján történik meg a *TmdbScore* becslése. Mivel ezeknek az adatoknak a típusa egész szám, ezért átalakításra van szükség: A *DataKind.Single* segítségével az ML.NET által a becslésekhez támogatott lebegőpontos számokká lehet őket alakítani, így jönnek létre a *KeywordFloat* és a *GenreFloat* mezők. Ugyan a *TmdbScore* értékének előrejelzésére irányul a becslés, viszont az adatok betanításához ennek értékeire is szükség van, viszont itt átalakítani nem szükséges, hiszen magának a bemeneti adatnak alapból float a típusa.

Az átalakított adatokkal létrejött új mezőket, a *KeywordFloat*-ot és a *GenreFloat*-ot végül egy mezőben, a *Features*-ben egyesítjük, ez fogja a bementi adatokat jelenteni a modell tanításához.

### 2.8.4 Algoritmus kiválasztása

Miután az adatokat megfelelő formába alakítottuk, kiválaszthatjuk a használni kívánt algoritmust attól függően, hogy mi a programunk célja. Az ML.NET függvénykönyvtárból több mint 30 algoritmus közül választhatunk. [27] Ugyanazon probléma megoldására több algoritmus is alkalmas lehet, így ahhoz, hogy megtaláljuk az adott helyzetben a legjobban működőt, érdemes kipróbálnunk minél többet.

Az ML.NET által támogatott algoritmusok között megtalálhatóak a következő típusú metódusok: [29]

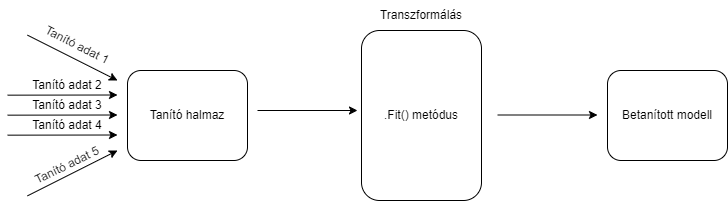
* Lineáris algoritmusok: A lineáris algoritmusok egy olyan modellt hoznak létre, amely a bemeneti adatok jellemzőit súlyozva és összegezve számítja ki a kimeneti értéket, ami ebben az esetben egy pontszám. A modell betanítása előtt normalizálni kell a jellemzőket ezen algoritmusok esetében, mivel így megakadályozható, hogy egyes jellemzők jobban befolyásolják a model működését mint a többi. [29]
* Döntési fa algoritmusok: Általánosságban véve rendkívül pontos eredményeket lehet velük elérni, döntések sorozatát tartalmazzák a kimeneti érték megállapításához. Működésükhöz a többi algoritmushoz viszonyítva nagyobb erőforrást vesznek igénybe. A lineáris algoritmusokhoz viszonyítva nagy különbségük, hogy a jellemzőket nem kell normalizálni. [29]
* Mátrix-faktorizációk: A collaborative filtering rendszerekhez használatos algoritmusok. [29] A collaborative filtering, vagy magyarul együttműködő szűrő rendszerek egy személynek különbőző más személyek, tárgyak, vagy információk iránti vonzalmát jelzik előre annak segítségével, hogy összevetik az adott személynek az érdeklődési körét az emberek többségének érdeklődési körével, így megkapják, hogy az adott személynek mi fog tetszeni az alapján, hogy más, hasonló ízlésű emberek mit kedveltek. [30]
* Meta-algoritmusok: Ezen algoritmusok segítségével bináris osztályozókból többosztályos osztályozók készíthetőek. [29]
* K-közép: A klaszterezés (beleértve a K-Közép) egy felügyelet nélküli tanítási mód. [31] [29]
* Fő komponens analízis: Hibakeresésre, anomáliák felderítésére használatos algoritmusok. [29]
* Naiv Bayes osztályozó: Többosztályos osztályozók esetében érdemes használni ezen algoritmusokat abban az esetben, mikor a jellemzők függetlenek egymástól, illetve kis mennyiségű adat áll rendelkezésünkre a model betanításához. [29]
* Prior trainer: A korábbi algoritmusokhoz képest ez egy egyszerűbb osztályozási problémára ad megoldást, hiszen csak bináris osztályozáshoz használhatóak. Sok esetben ilyen típusú algoritmusokat futtatnak először, mivel ez már ad egy viszonyítási értéket. Utána ha futtatunk más, különböző algoritmusokat, akkor azoktól ennél már jobb eredményt várunk el. [29]
* Vektorgépek támogatása (SVM): Felügyelt gépi tanulási algoritmusok, velük kapcsolatos megoldandó probléma, hogy hogyan lehet őket hatékonyan használni nagyobb adathalmazok esetében. [29]
* Legkisebb négyzetek módszere: A lineáris regressziónak ez egyik leggyakrabban használt felhasználási módszere. Veszteségfüggvénnyel méri a modell hibáját, azaz azt, hogy mennyire tér el a modell által előrejelzett érték a ténylegesen megfigyelt értéktől. [29]

### 2.8.5 Modell tanítása

A fentiek közül bármelyik típusú algoritmusból is választunk, ténylegesen csak akkor kerül végrehajtásra, ha meghívjuk a tanító adatok halmazára a .Fit() metódust. Ez az a pont ahol a modell tanítása, tehát maga a paraméterbecslés, a modell illesztése és hangolása történik. [29]

A modell betanításának folyamatát a 8. ábra szemlélteti. A folyamatban a tanító adatok lesznek a bemeneti adatok, majd a modell tanítása, mint egy transzformátor jelenik meg. A transzformálás után kimenetként a betanított modellt kapjuk, amellyel előrejelzéseket, becsléseket készíthetünk a még ismeretlen adatokra vonatkozóan. [29]

ITransformer model = pipeline.Fit(trainingData);



**8.** ábra: A modell betanításának a folyamata ML.NET-ben

### 2.8.6 Modell kiértékelése

A modellünk kiértékelésénél először azt kell figyelembe vennünk, hogy milyen típusú algoritmust használtunk a modell betanításához. Más-más módszerekkel kell kiértékelnünk a bináris osztályozást megvalósító modellt, a többosztályos osztályozát megvalósító modellt, a regressziós modellt, a klaszterezést megvalósító modellt, illetve a rangsorolást, anomáliaészlelést és a mondatok hasonlóságának vizsgálatát megvalósító modelleket is.

A szakdolgozatomban a filmajánlórendszernél többosztályos osztályozási probléma, illetve regressziós osztályozási problémák jelentek meg, így ezek értékelésére térek ki részletesen.

A többosztályos osztályozó esetében az alábbi értékek használhatóak a kiértékelésre: [32]

* Mikro-pontosság: A mikro-pontosságot (vagy többosztályos pontosságot) úgy határozhatjuk meg, hogy a helyesen megjósolt példányokat viszonyítjuk a teljes adathalmazhoz, tehát ezzel a módszerrel kapunk egy olyan arányszámot, ami a helyesen megjósolt példányok hányadát jelenti. [32] A mikro-pontosság minden esetet egyenlően kezel függetlenül az osztálytól, ezáltal a modell teljes teljesítményének vizsgálatában játszik nagyobb szerepet, hiszen egy egész, átfogó képet kaphatunk vele a modell pontosságáról. [33] A kapott érték minél közelebb van 1-hez, annál jobb, hiszen annál pontosabb a vizsgált modell. [32]
* Makro-pontosság: A mikro-pontossággal ellentétben a makro-pontosság osztályszinten vizsgálva jelenti az átlagos pontosságot. Tehát az osztályok pontosságait egyenként kiszámítjuk, majd magát a makro-pontosságot ezeknek a kiszámolt pontosságoknak az átlagaként kapjuk meg. Ezáltal a makro-pontosság mérésénél minden osztály, az osztályok gyakoriságának figyelembevétele nélkül egyenlően járul hozzá a végleges pontosság meghatározásához, pl. 20 osztály esetében minden egyes osztály 1/20-addal befolyásolja a kapott értéket. A mikro-pontossághoz hasonlóan itt is a kapott érték minél közelebb van 1-hez, annál jobb, hiszen annál pontosabb az aktuálisan vizsgált modell. [32]
* Log-veszteség: Az osztályozási modell teljesítménye olyan formában mérhető vele, hogy a log-veszteség értéke annál jobban növekszik, minél jobban eltér a megjósolt eredmény a ténylegestől. Tehát minél kisebb a los-veszteség értéke, a modellünk annál pontosabb becslésre képes. [32]
* Log-veszteség csökkentése: A log-veszteség csökkentésénél a véletlenszerű találhatáshoz viszonyítunk. Tehát a modellünket azzal hasonlítjuk össze hogy az adott modell mennyivel működik hatékonyabban annál, mintha csak véletlenszerűen megpróbálnánk az értékeket eltalálni. A kapott szám mínusz végtelen és 1 közé kell hogy essen, és minél nagyobb ez a szám, annál hatékonyabb a modellünk. Ha pl. 0,1 a kapott érték, akkor 10%-kal működik jobban a modell a véletlenszerű találgatásnál. [32]

A Microsoft ML.NET keretrendszerhez kapcsolódó, a modell kiértékeléséről szóló dokumentációja [32] szerint ha a felsorolt hatékonyságmérési módszerek közül csak egyet lehetne választani, akkor a legtöbb esetben mindenképpen a mikro-pontosságot a legérdemesebb.

Az alábbi kódrészlet a mikro-pontosság kiszámítását és kiíratását mutatja be:

var predictions = model.Transform(data);

var metrics = mlContext.MulticlassClassification.Evaluate(predictions);

Console.WriteLine($"MicroAccuracy: {metrics.MicroAccuracy}");

Regressziót használó osztályozók esetében az alábbi módszerek használhatóak a kiértékelésre: [32]

* R-négyzet: Annak a mérőszáma, hogy a becsült értékek mennyire egyeznek meg a tesztadatokkal. A kapott érték mínusz végtelen és 1 között lehet. Ezt a tartományt 3 részre tudjuk osztani: ha 0 és 1 között van az érték, akkor a modell hatékonyabban működik a véletlen találgatásnál, minél nagyobb ez a szám, annál pontosabb. Ha pont 0 az értéke, akkor megegyezik a véletlenszerű találgatással. Mínusz érték esetén pedig rosszabbul működik a betanított modell, mintha egyszerűen véletlenszerűen találgatna.
* Abszolút veszteség: Szintén arra vonatkozó mérőszám, hogy a becsült értékek mennyire állnak közel a valóshoz. Minden egyes értékre kiszámításra kerül a hiba mértéke, amely a becsült érték és a tényleges közötti abszolút távolság. Végül ezekből az értékekből számolunk egy átlagot, ez lesz a kapott abszolut veszteség. Minél közelebb van a szám a 0-hoz, annál pontosabb a modell.
* Négyzetes eltérés: A regressziós egyenes és a tesztadatok értékkészletének kapcsolatát vizsgálja úgy, hogy a pontok és az egyenes közötti távolságokat négyzetre emeli. A négyzetre emelés miatt a nagyobb távolságok nagyobb súlyt kapnak. A kapott érték minden esetben 0 vagy pozitív, viszont minél közelebb van a 0-hoz, annál jobban működik a modell.
* RMS-veszteség: A négyzetes eltérés négyzetgyöke, ezáltal könnyebben értelmezhetővé teszi a hibamérést. Minél közelebb van a 0-hoz, annál pontosabb a vizsgált modell.

### 2.8.7 A modell kimentése, használata

Az elkészített modellek kimentésére is lehetőséget kínál az ML.NET keretrendszer. Alapesetben a program minden futáskor új modellt generál az adott, aktuális jellemzők alapján, ez a memóriában kerül tárolásra addig a pontig, ameddig a program fut. Ha az elkészített modellünket a későbbiekben is szeretnénk használni, tesztelni, esetleg más programokban felhasználni, akkor kimenthetjük azt egy .zip kiterjesztésű tömörített fájlba. [34]

A modell mentéséhez a Microsoft ML.NET dokumentációja [34] szerint 2 dologra van szükség:

* Az elkészített modell ITransformerére, tehát gyakorlatilag a program azon részére, ahol a modellt betanítottuk a tanító adatokkal

var model = trainingPipeline.Fit(data);

* Az ITransformer várható bemenetének a DataViewSchema-ja, tehát a bemeneti adatok sémája

var data = mlContext.Data.LoadFromEnumerable(movies, schemaDef);

A modell kimentése a fenti változók segítségével:

mlContext.Model.Save(trainedModel, data.Schema, "model.zip");

(Az ML.NET dokumentációjából [34] származó kódrészlet)

A kimentett modell visszatöltéséhez egy DataViewSchema és egy ITransformer típusú változóra van szükségünk: [34]

DataViewSchema modelSchema;

ITransformer trainedModel = mlContext.Model.Load("model.zip", out modelSchema);

Az ITransformer típusú változóba betöltjük a korábban kimentett, modellt tartalmazó zip fájlt, és ezzel gyakorlatilag megkapjuk a modell sémáját is.

## 2.9 Algoritmusok hatékonyságának összehasonlítása

A szakdolgozatom alapját egy folytonos érték becslése adja, melynek segítségével lehetséges értékelni a filmeket abból a szempontból, hogy mennyire tetszene az adott felhasználónak. Így az ML.NET-ben elérhető, különböző becslő algoritmusokat fogok összehasonlítani hatékonyság szempontjából.

A pontosság mérését a MovieRecommendationSystem *MeasureAccuracy* nevű osztályában valósítottam meg. A hatékonyság méréséhez az R2 mérőszámot fogom használni, és a TMDB pontszám becslését megvalósító *DecTreeForTMDB* osztály kódját alakítottam át az egyes becslési módszerek esetén olyan formába, hogy az adathalmaz minden elemére lefusson a becslés, és alkalmas legyen a pontosság mérésére.

Az R2, mint a pontosság mérésére vonatkozó mérőszám már korábban említésre került, viszont az említett formában csak azt mutatta meg, hogy a betanítás után a modell mennyire pontosan tudja megbecsülni a betanító adatokat. Az adatbázisom 100 elemű, viszont a becslésekhez ennek csak egy részét használtam fel, így létrehoztam egy külön, a modellek R2 pontosságának mérésére szolgáló metódust annak a problémának a kiküszöbölésére, hogy ne csak azt lehessen megnézni, hogy a tanító adatokon milyen pontossággal működik, hanem azt is hogy az egész adatbázis adataira hogyan működik. A becslések a filmek TMDB pontszámára vonatkoztak, ami minden film esetén ismert az adatbázisban, ezért könnyen össze lehetett őket vetni a becsült értékekkel.

A vizsgálat szempontja tehát a következő volt: A 100 darab film TMDB pontszámának, tehát 1 és 10 közötti lebegőpontos értékek megbecslése egy 20 elemű tanítóhalmaz alapján.

A kapott eredmények R2 pontosságai a 3. táblázatban láthatóak.

**3.** táblázat: A különböző algoritmusok mérései során kapott R2 értékek

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1.** | **2.** | **3.** | **4.** | **5.** |
| Fast tree | -69,63 | -69,63 | -69,63 | -69,63 | -69,63 |
| Fast tree Tuned | 0,172 | 0,128 | 0,162 | 0,117 | 0,184 |
| Sdca | 0,07 | 0,08 | 0,11 | 0,098 | 0,103 |
| Sdca Tuned | 0,143 | 0,145 | 0,148 | 0,14 | 0,152 |
| Lbfgs Poisson R. | -0,17 | -0,17 | -0,172 | -0,17 | -0,17 |
| Lbfgs Poisson R. Tuned | -0,190 | -1,822 | -0,163 | -1,586 | -0,846 |
| Gam | -69,63 | -69,63 | -69,63 | -69,63 | -69,63 |
| Gam tuned | -69,63 | -69,63 | -69,63 | -69,63 | -69,63 |
| Online Gradient Descent | -15,32 | -11,29 | -12,342 | -8,227 | -14,286 |
| Online Gradient Descent tuned | -5,74 | -5,716 | -5,08 | -4,399 | -5,139 |

Minden algoritmust lefuttattam úgy, hogy a modellt különböző paraméterekkel megpróbáltam finomhangolni, illetve ezen paraméterek nélkül is. A finomhangolás során véletlen számokat generáltam bizonyos tartományokon belül a különböző paraméterekhez. A tartományoknál megpróbáltam figyelembe venni, hogy 20 elemszámú tanító halmazom van, tehát viszonylag kis méretű, és ehhez viszonyítva próbáltam az alsó és felső határokat meghatározni. Mindegyik algoritmusnak a finomhangolt verziójához 200 alkalommal generáltam véletlen számokat minden egyes parameter esetén, ezzel próbáltam megkeresni azokat a parameter értékeket, amelyekkel a leghatékonyabban működik a modell, és az ezekhez tartozó legmagasabb R2 értékek kerültek a végén megjelenítésre, illetve a táblázatba.

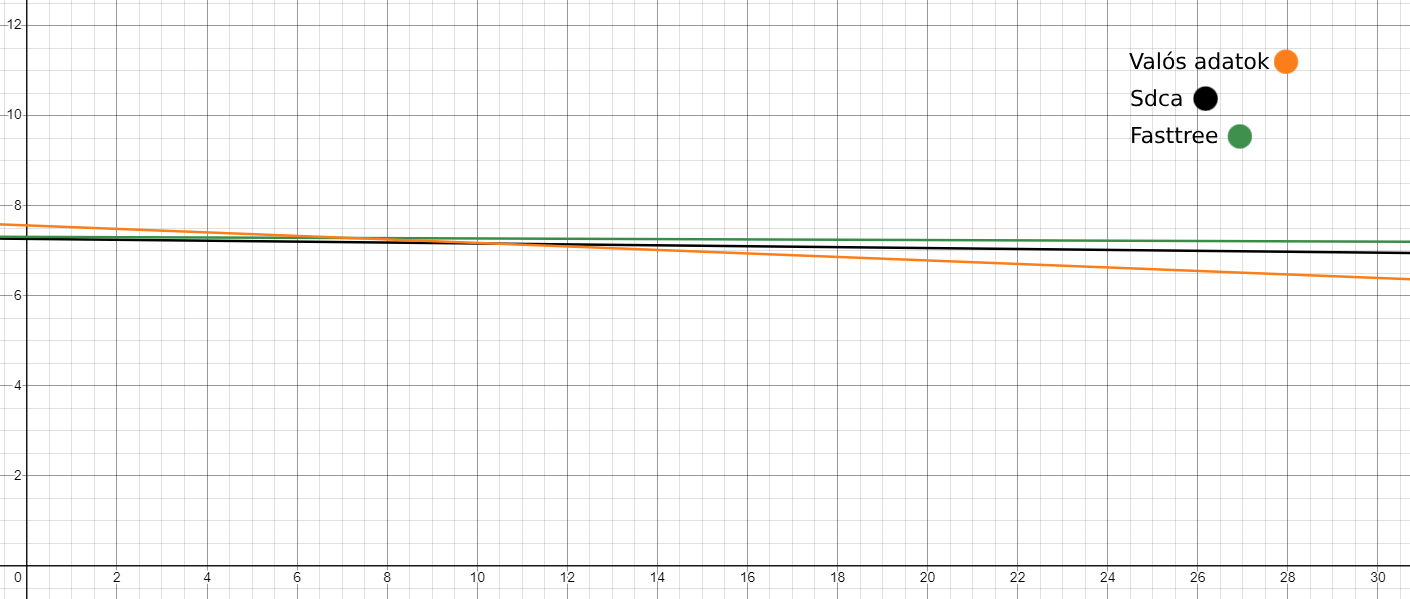
Az R2 mérőszám értéke mínusz végtelentől 1-ig terjedhet. Minél magasabb az érték, tehát minél közelebb van 1-hez, annál hatékonyabban működik a modell. Ha negatív értéket vesz fel, az azt jelöli, hogy a modell rosszabbul működik mint a véletlen találgatás. Ezen információk ismeretében az alábbi következtetések mondhatóak el a mérések alapján:

Finomhangolás nélkül egyetlen algoritmus tudott jobb eredményt hozni, mint a véletlen találgatás, ez pedig az Sdca volt. Az összes többi algoritmus még rosszabbul teljesített, mint a véletlen találgatás.

Finomhangolással az LbfgsPoissonRegression-ön és a Gam algoritmuson kívül mindegyiken lehetett javítani, és jobb értékeket lehetett elérni, mint finomhangolási paraméterek használata nélkül, viszont csak a Fasttree algoritmus tudott a véletlen találgatásnál jobb értékeket hozni.

Finomhangolás után a Fasttree és az Sdca algoritmusok közel azonos pontosságot hoztak, az 5 darab mérés eredményéből átlagot számítva az Sdca esetén 0,1545-et, a Fasttree esetén 0,1691-et kapunk, tehát az R2 mérőszámokat figyelembe véve a Fasttree algoritmus működött az összes közül a leghatékonyabban.

A két legjobban működő algoritmus becsléseit lineáris regresszió használatával ábrázoltam a valós adatok mellett, ezt a 9. ábra szemlélteti.



**9**. ábra: A Fasttree és Sdca becslései lineáris regresszióval ábrázolva a valós adatok mellett

Egy 30 filmből álló tartomány értékeit ábrázolja a lineáris regresszió. Mint az ábra is mutatja, narancssárga színnel van ábrázolva a valós adatok egyenese, fekete színnel az Sdca algoritmus által becsült adatok egyenese, és zöld színnel pedig a Fasttree algoritmus által becsült értékek egyenese. Az ábrából látszik, hogy a két algoritmus becslései közel hasonló meredekséget követnek, itt is fellelhető közöttük a működésbeli hasonlóság, mint ahogyan az R2 mérőszámmal végzett pontosság mérésénél is.

A fenti eredmények alapján kijelenthető, hogy az említett problémára a Fasttree algoritmus adta a legpontosabb becsléseket az adott paramétertartományokban megtalált legjobb paraméterértékekkel. Viszont a 2 algoritmus eredményei rendkívül közel vannak egymáshoz, szóval a tartományok határával való további kísérletezéssel, vagy akár a tanítóhalmaz elemszámának növelésével/csökkentésével változni fog a pontosság, így akár az is, hogy melyik algoritmus teljesít jobban a másiknál.

# 5. Irodalomjegyzék

[1] <https://www.inf.u-szeged.hu/~rfarkas/ML22/embedding.html>

[2]: <https://link.springer.com/article/10.1007/s10462-019-09744-1>

[3]: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=199700bb1e2329b973c96b019d8c7d6910e6911a>

[4]: <https://www.irjet.net/archives/V3/i8/IRJET-V3I8393.pdf>

[5] <https://dl.acm.org/doi/pdf/10.1145/234313.234346>

[6] <https://cdn.aaai.org/AAAI/2006/AAAI06-080.pdf>

[7] <https://www.sqlite.org/>

[8] [https://www.kaggle.com/datasets/TMDB/TMDB-movie-metadata](https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata)

[9] <https://www.themoviedb.org/>

[10] <https://www.researchgate.net/profile/Pramila-Chawan/publication/316514673_A_Comparative_Study_of_Keyword_and_Semantic_based_Search_Engine/links/5901cfc7a6fdcc8ed51110da/A-Comparative-Study-of-Keyword-and-Semantic-based-Search-Engine.pdf>

[11] <https://www.diva-portal.org/smash/get/diva2:1088795/FULLTEXT01.pdf>

[12] <https://www.jatit.org/volumes/Vol96No6/4Vol96No6.pdf>

[13] <https://www.sciencedirect.com/science/article/abs/pii/S1570826815001432>

[14] <https://link.springer.com/chapter/10.1007/978-3-540-72667-8_26>

[15] <https://www.researchgate.net/profile/K-Kranthi-Kumar-2/publication/235634738_CBIR_Content_Based_Image_Retrieval/links/004635189dd9ac1e6f000000/CBIR-Content-Based-Image-Retrieval.pdf>

[16]: <https://www.sqlite.org/index.html>

[17]: <https://www.sqlite.org/whentouse.html>

[18]: <https://survey.stackoverflow.co/2024/technology#most-popular-technologies-database>

[19]: <https://system.data.sqlite.org/index.html/doc/trunk/www/index.wiki>

[20]: <https://medium.com/@adnankattekaden/exploring-the-evolution-benefits-and-limitations-of-json-a-beginners-guide-7d8ba5e1c5e4>

[21]: <https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning>

[22]: <https://www.mlpack.org/>

[23]: <https://scikit-learn.org/stable/>

[24]: <https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet>

[25]: <https://en.wikipedia.org/wiki/ML.NET>

[26]: <https://learn.microsoft.com/hu-hu/dotnet/machine-learning/how-does-mldotnet-work>

[27]: <https://dotnet.microsoft.com/en-us/learn/ml-dotnet/what-is-mldotnet>

[28]: <https://learn.microsoft.com/en-us/dotnet/machine-learning/resources/transforms?WT.mc_id=dotnet-35129-website>

[29]: <https://learn.microsoft.com/en-us/dotnet/machine-learning/resources/tasks?WT.mc_id=dotnet-35129-website>

[30]: <https://dl.acm.org/doi/abs/10.1145/358916.358995>

[31]: <https://www.sciencedirect.com/science/article/abs/pii/S0950705106000189>

[32]: <https://learn.microsoft.com/en-us/dotnet/machine-learning/resources/metrics?WT.mc_id=dotnet-35129-website>

[33]: <https://vitalflux.com/micro-average-macro-average-scoring-metrics-multi-class-classification-python/#What_Why_of_Micro_Macro-averaging_and_Weighting_metrics>

[34]: <https://learn.microsoft.com/en-us/dotnet/machine-learning/how-to-guides/save-load-machine-learning-models-ml-net?WT.mc_id=dotnet-35129-website>