Szegedi Tudományegyetem

Informatikai Intézet

Ajánlórendszer algoritmusainak megvalósítása és vizsgálata

Szakdolgozat

Készítette:

Dodony Róbert

informatika szakos

hallgató

Témavezetõ:

Dr. Dombi József

professzor emeritus

Szeged

2019

Feladatkiírás

Az ajánlórendszerek kettős szerepet töltenek be az üzleti életben. Egyrészt a vásárlók figyelmét felhívják a vásárlásuk alapján, hogy még milyen egyéb árut lenne érdemes megvenniük, másrészt a kereskedők a beszerzéseiket az ajánlórendszerek által nyújtott információkra is alapozhatják. Az online vásárlások egyre nagyobb mértékben való elterjedése miatt az ajánlórendszerek webes környezetben különösen nagy jelentőséggel bírnak. A hallgató feladata egy pozitív értékeléseken alapuló ajánlórendszer implementálása. Az algoritmusok megismerése után egy minta adatbázison futtatva implementálja a programot egy webes rendszerbe.

Tartalmi összefoglaló

– *A téma megnevezése:*

  Pozitív értékelésekre alapuló ajánlórendszer.

– *A megadott feladat megfogalmazása:*

  Olyan algoritmus megvalósítása és vizsgálata, amely egy unáris értékelési halmazt felhasználva tud ajánlatokat tenni.

– *A megoldási mód:*

  SLIM algoritmus megvalósítása Java nyelven.

– *Alkalmazott eszközök, módszerek:*

  Windows 10 operációs rendszeren, Eclipse JEE fejlesztői környezet, Java 1.8 verzió, MySQL adatbázis, Spring Boot 2.0+, Hibernate 5.4+, Maven. A fejlesztés nyomon követéséhez a Github verziókövető rendszer alkalmazása.

– *Elért eredmények:*

  Pozitív értékeléseket alkalmazó ajánlórendszer megtervezése. A SLIM algoritmus megvalósítása, úgynevezett ”feature selection”-nel való bővítése, ”ridge-” és ”lasso” regresszió bevezetése. Az ajánlórendszer implementálása webes környezetbe.

– *Kulcsszavak:*

  Ajánlórendszer, Termék alapú, Unáris értékelés, SLIM, Java, Spring boot, Hibernate.

Tartalomjegyzék

Feladatkiírás 2

Tartalmi összefoglaló 3

Bevezetés 6

1. Az ajánlórendszerek fejlődése **7**

1.1. Implicit és explicit értékelési módszerek 8

1.2. Alapfogalmak és jelölések 9

1.3. Az ajánlórendszerek funkciói és céljai 9

1.4. Értékelések típusai 10

1.5. Az úgynevezett ”Long tail” probléma 11

2. Kollaboratív szűrésen alapuló modellek **12**

2.1. Felhasználó alapú szomszéd modell 13

2.2. Termék alapú szomszéd modell 14

2.3. Regressziós modellek használata szomszéd alapú módszereken 15

2.4. Felhasználó alapú legközelebbi szomszéd regressziós módszer 16

2.5. Termék alapú legközelebbi szomszéd regressziós módszer 18

3. Ritkasági lineáris modellek (SLIM – Sparse linear models) **20**

3.1. SLIM jelölések 21

3.2. SLIM működése 22

3.3. W kiszámolása 23

3.4. SLIM ”feature selection” 24

4. Megvalósításhoz használt technológiák **25**

4.1. Java 25

4.2. Eclipse 25
4.3. Spring Boot 26
4.4. Thymeleaf 26
4.5. Maven 26
4.6. MySQL 26
4.7. Git 27

5. A SLIM algoritmus megvalósítása **28**
5.1. SimilarityCalculator 28
5.2. DerivationHelper 29
5.3. ExpressionHelper 30
5.4. GaussEliminationHelper 30
5.5. RecommenderHelper 31
5.6. VectorHelper 31
5.7. Recommender 32
5.8. Program felépítése 33
5.9. Tesztelés 33

Összegzés **36**

Irodalomjegyzék **37**
Mellékletek 38

Nyilatkozat 40
Köszönetnyilvánítás 41

Bevezetés

Az ajánlórendszerek lehetővé teszik a felhasználók számára információ- és terméklisták szűrését. Az elmúlt 3 évtizedben sok új algoritmust és eszköszt fejlesztettek ki a feladatok megoldására. Mivel minden az ígények különböznek, ezért nincs minden helyzetre alkalmas ajánlórendszer sem.

Minden nap új döntéseket kell hoznunk. Mit vegyünk fel? Mit együnk? Melyik könyvet vegyük meg? Az emberek a külső forrásokra (ismerősök, médiumok stb.) hagyatkoztak, ez azonban egy korlátozott ismerethalmazt jelentett. Előfordulhat, hogy a felhasználó igényeinek megfelelne egy könyv, azonban erről a termékről való információ nem jut el hozzá. Az ajánlórendszerek nagy előnye, hogy ezt az ”ismeretségi kört” kibővítik az adott rendszert használó összes személyre, így a felhasználó nincs korlátozva az őt érő információtól.

Az ajánlórendszer algoritmusának célja a felhasználó viselkedésének az elemzése és ezek alapján megfelelő, személyre szabott javaslatok megtétele, ami azonban az egész problémának csupán egy része, ugyanis egy megfelelő grafikus felhasználói felület is szükséges, amely értelmezhetővé teszi a rendszer eredményét, vagyis a felület továbbra is felhasználóbarát marad, de mégis tud elég információt kinyerni a felhasználóról. Fontos a felhasználók privát szférájának sértetlenségének a megőrzése.

A 3. fejezet a [1, 2] forrás alapján készült.

1. fejezet

Az ajánlórendszerek fejlődése

Az első ilyen irányú előrelépés Grundy nevéhez fűződik, cikkében a felhasználókat rövid leírások alapján kategóriákba (sztereotípiákba) sorolta. Az 1990-es években jelent meg a kollaboratív szűrés. A Tapestry szűrési eljárás egy kifejlesztett manuális kollaboratív szűrőrendszer volt, ami a felhasználók számára lehetővé tette, hogy lekérdezések alapján szűrjék az információt (pl. céges e-maileket). Ezt követték az automata kollaboratív szűrőrendszerek, az első ilyen a GroupLens volt, amely a Usenet olvasók értékelései alapján előre megjósolta, hogy egy adott felhasználónak tetszene-e egy új cikk (mielőtt elolvasta volna). További jelentős ajánlórendszerek Ringo (zenék), BellCore Video Recommender (filmek), Jester (viccek).

A marketing is komoly érdeklődést mutatott az ajánlórendszerek iránt, mivel növelik az eladások számát. Az 1990-as évek végén megjelentek az első kereskedelmi ajánlórendszerek is. Legismertebb ezek közül az Amazon.com, amely vásárlási történet, böngészési történet és a felhasználó által éppen meglátogatott termék oldala alapján kínál további termékeket. Az ajánlórendszer implicit és explicit értékeléseket is figyelembe vesz. Implicit értékeléseket a felhasználó ad meg, például egy 1 és 5 közötti skálán. Explicit értékeléseket a felhasználó viselkedéséből szűrhetünk le, például megvásárolt vagy megtekintett egy terméket. Az ajánlások részleteses magyarázattal bővítettek, amely a felhasználónak nyújthat információt a kínált termék hasznosságáról.

Netflix – Filmeket és sorozatokat kínál a felhasználóknak a már előzőleg megtekintett és értékelt termékek alapján. Az ajánlásokhoz magyarázatot is készít, rövid leírást ad arról, hogy miért is lehetne ízlésére a felhasználónak (pl. : nézett egy hasonló kategóriájú filmet,

amire magas értékelést adott).

Google news - felhasználók böngészési történetét használja egyéb releváns cikkek keresésére. Az oldalak látogatásából leszűrt információkat a Gmail felhasználókhoz rendeli és a megtekintett cikkeket tekinti a termékeknek. Ha a felhasználó megtekint egy cikket, akkor ezzel ”tetszést” fejez ki az adott cikkre. Ez unáris értékelési rendszert használ, ahol csak a tetszés kifejezésére van lehetőség, a felhasználó nem tudja jelölni, hogy egy cikk nem tetszik neki.

Az ajánlórendszerek két alapvető modellre bonthatók, első a felhasználó-termék viszonyon alapuló, második attribútum információ alapú (felhasználóra és tárgyra is vonatkozhat). Az első típusú a collaborative filtering, a második a content-based recommenders.

1.1. Implicit és explicit értékelési módszerek

Implicit értékelési rendszernek nevezzük az olyan ajánlórendszereket, amelyek egy tevékenységből leszűrik egy termékre vonatkozó értékeléssel kapcsolatos információkat (pl.: egy 5 csillagos értékelési lehetőség esetén értékelésünkkel meghatározhatjuk a termék iránti affinitásunkat). Explicit értékelésnek nevezzük, amikor a felhasználó megvásárol vagy megnéz egy terméket, ezzel jelezve, hogy valamilyen szinten érdekli a termék.

Az implicit és az explicit módszereket az ajánlórendszer együttesen használja, hogy személyre szabott javaslatokat tegyen a felhasználóknak. Tehát az ajánlórendszerek múltbéli felhasználó-tárgy viszonyokból szűrik ki a megfelelő információt. Pl.: ha egy felhasználónak tetszett egy drámai könyv, akkor nagyobb eséllyel fog neki tetszeni egy másik ugyanilyen könyv, mint egy történelemi könyv. Az értékelések alapján különböző műfajok értékelései korrelációt mutathatnak, így más kategóriájú termékeket is javasolhat az ajánlórendszer, ekkor az előző példát tekintve a történelmi könyv is megfelelhet az igényeinek. Minél több terméket értékelt egy felhasználó, annál pontosabb értékeléseket becsülhetünk az általa még nem értékelt tárgyakra.

1.2. Alapfogalmak és jelölések

Egy kollaboratív szűrő rendszer információ tartománya olyan felhasználókból áll, amelyek értékeltek bizonyos tárgyakat. A felhasználónak a tárgy iránti előnyben részesítésének a mértékét nevezzük értékelésnek (rating) nevezzük, és ezt egy (Felhasználó (User), Tárgy (Item), Értékelés (Rating)) hármas jelöli.

Az összes értékesítési hármas halmaza adja az értékelési mátrixot (ratings matrix), amelyet **R** jelöl. Az **R** egy $m \times n$ mátrix, ahol m a felhasználók száma, n pedig a tárgyak száma. Ez a mátrix tartalmaz (Felhasználó, Tárgy) párokat is, amelyek azt jelölik, hogy a felhasználó még nem értékelte a tárgyat.

Jelölje U a felhasználók (users) halmazát, I pedig a termékek (items) halmazát. Jelöljön egy felhasználót u, egy terméket i és egy értékelést r. Jelölje I_u az u felhasználó által értékelt tárgyak halmazát. Jelölje U_i azon felhasználók halmazát, akik értékelték i tárgyat. Jelölje $r_{u,i}$ az u felhasználó által értékelt i tárgy, r_u az u felhasználó értékeléseinek a vektorát, r_i pedig az i tárgyra adott értékelések vektorát. Jelölje μ_u az u felhasználó értékeléseinek az átlagát, μ pedig az i tárgyra adott értékelések átlagát.

1.3. Az ajánlórendszerek funkciói és céljai

Két fő modellre bontható az ajánlórendszerek működése:

1. Hiányzó értékelések becslése: Az m (felhasználók) és n (termékek) által meghatározott $m \times n$ -es **R** mátrix hiányzó értékeinek a becslése. Feltételezzük, hogy "training data" rendelkezésünkre áll (a már meglévő értékelések). Ez alapján becsüljük meg a mátrixban lévő hiányzó értékeléseket. Az adott probléma "matrix completion problem" néven is ismert.

2. Rangsorolás: Ehhez nem szükséges a hiányzó értékelések becslése, mert előfordulhat, hogy az eladó a legjobb k darab (top-k) terméket szeretné javasolni egy bizonyos felhasználónak. A probléma "top-k recommendation problem" néven is ismert.

Az ajánlórendszerek alapvető célja az eladások számának növelése, ennek eléréséhez az általános működési és technikai céljai a következők:

1. Jelentőség: A legalapvetőbb működési elv egy ajánlórendszernél, hogy a felhasználó számára fontos termékeket ajánljon.

2. Újdonság: Az ismételt javaslatok az eladások számának a csökkenéséhez vezethet, ezért fontos, hogy a felhasználó számára változatos ajánlatokat tegyen a rendszer, olyan termékeket kínáljon, amelyekkel még nem találkozott.

3. Szerencsés véletlen: Olyan termékek ajánlása, amely a felhasználó számára is meglepő. Ez annyiban különbözik az újdonságtól, hogy a felhasználó számára teljesen jelentéktelen is lehet. A felhasználó megismeréséhez alkalmazott technika, mivel egy eddig nem értékelt dologról vonhatunk le új információt, azzal kapcsolatban, hogy érdekli-e az adott termék.

4. Változatosság: Az ajánlórendszer gyakran egy top-k listát ajánl a felhasználónak, azonban ez sokszor nagyon hasonló termékeket tartalmaz. Ekkor fennáll a kockázata annak, hogy a felhasználó megunja ezt az adott termékkategóriát. Ezért hasznos egyszerre változatos kategóriájú termékeket ajánlani.

1.4. Értékelések típusai

Az értékelési skálák a ”nem tetszés” -től a ”tetszés” -ig terjednek. Típusuk jelentősen befolyásolhatja az alkalmazott ajánlórendszer algoritmust. A következők fordulnak elő:

– Folyamatos skálák, két véglet között bármilyen értéket felvehet (pl.: a Jester ilyet alkalmaz, ahol az értékelés -10 és 10 között bármilyen értéket felvehet). Hátránya hogy ”nehéz” döntés elé állítja a felhasználót, hogy végtelen lehetséges pontozás közül válasszon egyet.

– Intervallum alapú skálák, legelterjedtebb a 5, 7, illetve 10 pontos (pl.: egy 5 pont alapú 1, 2, 3, 4, 5 értékeket tartalmaz), fontos megjegyezni, hogy mindegyik érték egyenlő távolságra van a szomszédjaitól (általában). Az értékek jelentése nem mindig triviális, a Netflix értékelési skáláján a 4 = nagyon tetszett, 3 = tetszett, így három darab pozitív és kettő darab negatív értékelésből áll. Az ilyen módszert erőltetett döntés értékelési rendszernek (forced choice rating system) nevezzük.

– Sorszám alapú értékelések kategóriákból állnak (pl.: strongly disagree, disagree, neutral, agree, strongly agree), itt nem feltétlenül azonos különbségek vannak az opciók között (de sokszor gyakorlatban ezek az értékek is 1-5 skálán lévő egész értékű számnak felelnek meg).

– Bináris értékelési rendszernek nevezzük, amikor a felhasználónak két opciója van az értékelésre ”tetszik” és ”nem tetszik”, ilyent használ például a Youtube.

– Az értékelési rendszerek egy speciális esete az unáris rendszer, ami csak a “tetszés” opcióból áll. Ez főleg implicit adathalmazoknál használt, ha a felhasználó megvásárol egy terméket, akkor ezzel ”tetszés” -t fejez ki, viszont ha nem vásárolja meg egy terméket, akkor az nem feltétlenül jelent ”nem tetszés” -t. Hasonló módon működik a Facebook is.

1.5. Az úgynevezett ”Long tail” probléma

A valóságban az értékelések eloszlása nem egyenletes, ugyanis általában csak néhány termék kerül gyakran értékelésre, a többi pedig igen ritkán. Ezeket a gyakran értékelt termékeket nevezzük a népszerű termékeknek. Ezt nevezzük a hosszú farok (long tail) tulajdonságnak. A népszerű termékek általában a legversenyképesebbek, ezért kis profitot termelnek az eladó számára, ezzel szemben a kevésbé keresettek nagyobb profitot termelnek, ezért az eladónak érdemes ezeket kínálnia. A nem népszerű termékek minősítésének a hiánya miatt nehéz pontos becsléseket számolni a hosszú farokban, ezért az ajánlórendszerek hajlamosak a populáris termékeket ajánlani a népszerűtlenek helyett. A szomszéd alapú rendszer a népszerű termékek értékeléseinek a gyakorisága miatt hajlamos ezek az értékek alapján számolni, ami azonban nem feltétlenül tükrözi a népszerűtlen termékeket.

A 1. fejezet a [1, 2] forrás alapján készült.

2. fejezet

Kollaboratív szűrésen alapuló modellek

A kollaboratív szűrés alapú modellek a felhasználók által megadott értékelések alapján működnek. Ennél a módszernél a legnagyobb kihívás a mátrix hiányossága. A felhasználók nagy része csak a legmenőbb termékekkel kerül kapcsolatba, ez nehézséget jelent mivel a többi termékről kevés információnk lesz. Az alapötlet lényege, hogy ha a és b felhasználók nagyon hasonlóak, és a felhasználónak tetszett x termék, akkor nagy valószínűséggel b-nek is tetszeni fog. Kétféle eljárás létezik:

1. Memória alapú (Memory-based method): Más néven szomszéd alapú kollaboratív szűrő algoritmus. Ezek voltak az első kollaboratív szűrő algoritmusok, amelyek a felhasználó-termék viszonyú értékelést a szomszédok alapján becsüli meg. Nagy előnyük, hogy könnyen implementálhatók és az ajánlások viszonylag könnyedén magyarázhatóak. Hátrányuk azonban, hogy nagyon hiányos mátrixoknál nem megbízhatóak, mivel kevés információ áll rendelkezésre. Itt is kétfajta van:

   a) Felhasználó alapú: Egy A célfelhasználóhoz hasonló felhasználók értékelései alapján tesz a rendszer javaslatokat az A személynek. Az alapötlet az A-hoz leghasonlóbb felhasználók megkeresése. Az A által még nem megadott értékelésekre becslést számolunk a hozzá hasonló felhasználók értékeléseinek súlyozott átlagát véve. Két felhasználó hasonlóságát az értékelési mátrixban megfelelő sorok összevetésével kapjuk.

   b) Termék alapú: Egy B cél termék, A felhasználó általi becsült értékeléséhez tudnunk kell a B-hez leghasonlóbb termékek S halmazát. Az S halmaz csak

az A felhasználó által értékelt termékeket tartalmaz. Az S halmaz alapján számítjuk ki, hogy A felhasználónak mennyire tetszene B termék.

2. Modell alapú (Model-based method): Gépi tanulás és adatbányászati technikákat alkalmaznak a hiányzó értékelések megbecslésére. Ilyenek például a döntési fák, szabály alapú modellek, Bayes módszer és latent factor modellek. Előnyük a memória alapú modellekkel szemben, hogy hatékonyságuk nem függ annyira a mátrix hiányosságától.

2.1. Felhasználó alapú szomszéd modell

A cél az A cél felhasználóhoz hasonló felhasználók megkeresése és az értékeléseik alapján az A nem megadott értékeléseire egy becslés számítása. Ehhez ki kell számítanunk az A felhasználó hasonlóságát a többi felhasználóhoz képest. Két felhasználó hasonlóságát a mindkettőjük által értékelt termékek alapján határozzuk meg. Itt kihívást jelenthet, hogy senki sem egyforma, ezért lehet, hogy az egyik személy átlagosan magas értékeléseket ad, míg a másik átlagosan alacsony értékeléseket, ilyenkor a súlyozott átlagot is figyelembe kell venni.

$I_u \cap I_v$ jelöli az u és v felhasználó által is értékelt termékek halmazát, ez alapján számítjuk a hasonlóságot. Az egyik módszer a két vektor közötti hasonlóság kiszámolására a Pearson korreláció együttható. μ_u jelöli az u felhasználó értékeléseinek az átlagát, amelyet a kövekezőképpen számoluk:

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \qquad \forall u \in \{1 \dots m\} \tag{2.1}$$

Ezek után a Pearson korreláció együttható az u és v felhasználók között:

$$Sim(u,v) = Pearson(u,v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{2.2}$$

A számítás egyszerűsítése érdekében az is elfogadható, ha az μ_u egyszer számoljuk ki, az összes u felhasználó által értékelt termékre, így minden alkalommal ugyanazt az

értéket használjuk. Ez előnyösebb lehet, ha kevés közös értékelés van a két felhasználó között.

A Pearson korreláció együttható alapján határozzuk meg az u célfelhasználóhoz leghasonlóbb felhasználók halmazát, ez minden termékre külön történik, mivel nem biztos, hogy a leghasonlóbb felhasználóknak minden u által nem értékelt termékre van megadott értékelésük.

s_{uj} jelöli a súlyozott átlag értékelést, amelyet u felhasználó tett j termékre, következőképpen számoljuk:

$$s_{uj} = r_{uj} - \mu_u \qquad \forall u \in \{1 \dots m\} \tag{2.3}$$

$P_u(j)$ jelöli az u cél felhasználóhoz leghasonlóbb felhasználókat, akik értékelték j terméket. Az u cél felhasználó j termékre számított becsült értékét $\hat{r}_{uj}$ jelöli, amit így számolunk:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |Sim(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |Sim(u,v)|} \tag{2.4}$$

2.2. Termék alapú szomszéd modell

A termék alapú modellek nagy mértékben hasonlítanak a felhasználó alapú modellekhez, abban különböznek hogy a hasonlóság mértékét a termékek között számolják, nem pedig a felhasználók között. Az értékelési mátrix oszlopait hasonlítjuk (egy oszlop egy terméknek felel meg), míg a felhasználó alapúnál a mátrix sorait vetjük össze. A megadott értékelések súlyozott átlagát veszük figyelembe, tehát egy termékre leadott összes értékelés átlagát kivonjuk az adott értékelésből.

Az adjusted cosine similarity két termék (oszlop) i és j között a következőképp számolható:

$$AdjustedCosine(i,j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s_{uj}^2}} \tag{2.5}$$

Adjusted cosine hasonlóság helyett Pearson korreláció is használható, azonban a termék alapú metódusoknál az adjusted cosine gyakran pontosabb eredményeket ad.

Ha t cél termékre szeretnék becslést adni u felhasználót figyelve, akkor először a t termékhez laghasonlóbb top-k tárgyakat kell megkeresni, ezt az adjusted cosine hasonlósággal tehetjük meg. A t-hez leghasonlóbb termékeket, amiket u felhasználó értékelt, jelöljük $Q_t(u)$-val. A t termékre tett u felhasználó becsült értékét így számoljuk:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} AdjustedCosine(j,t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |AdjustedCosine(j,t)|} \tag{2.6}$$

A becslés a felhasználó hasonló termékekre tett értékelését használja fel. Például azonos műfajú filmek értékelései alapján jó becslést tehetünk egy még nem értékelt filmre.

2.3. Regressziós modellek használata szomszéd alapú módszereken

A felhasználó- és a termék-alapú módszerek is lineáris függvényként tesznek becslést az értékelésére, ehhez a szomszédos felhasználók értékeléseit az adott termékre vagy a felhasználó által értékelt szomszédos termékek értékeléseit használják fel.

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |Sim(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |Sim(u,v)|} \tag{2.7}$$

Felhasználó alapú szomszédos modellnél a becsült értékelés egy súlyozott lineáris kombinációja az adott termékre tett más értékelésekből. A felhasználók halmaza korlátozott az adott u-hoz leghasonlóbb k számú felhasználókra, akik értékelték j terméket, ezt $P_u(j)$ jelöli. Ha nem csak a legközelebbi k felhasználót figyeljük, hanem az összes j-re tett értékelést figyelembe vesszük, akkor a becslés lineáris regresszióra kezd hasonlítani. Lineáris regressziónál a becslés szintén súlyozva van, a súlyozási értékeket egy optimalizálási modellel számítjuk ki. Szomszéd alapú módszereknél a súlyozási értékeket heurisztikus módon számítjuk a felhasználók közötti hasonlóság mértéke alapján.

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} AdjustedCosine(j,t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |AdjustedCosine(j,t)|} \tag{2.8}$$

Hasonló módon a termék alapú szomszédos modellnél $Q_t(u)$ jelöli a t-hez leghasonlóbb k darab tárgyat, amelyet u felhasználó értékelt. Itt u felhasználó t termékre tett becslését a már meglévő értékeléseinek a lineáris kombinációja adja. A felhasználó alapú módszer a mátrix egy oszlopában keresi a hasonlóságot, ezzel ellentétben a termék alapú pedig a mátrix egy sorát vizsgálja. Ez alapján a szomszéd alapú modell a lineáris regresszió egy olyan változata, ahol a regresszió együtthatóit a hasonlósági mérték határozza meg a szomszédos termékeknél/felhasználóknál, míg a jelentősen különbözőknél ez az érték 0 lesz. Ilyen esetben a hasonlósági együtthatók nem veszik figyelembe a termékek közötti kölcsönös függőségeket. Például, ha egy felhasználó hasonló termékeket hasonlóan értékelt, akkor ezek a termékek értékelése kölcsönösen függ egymástól, azonban ezzel a hasonlóságok által számolt súlyozás nem számol.

Felvethető a kérdés, hogy érdemesebb-e a súlyok kiszámolását egy optimalizálási feladatként kezelni? Regressziós modellek alakíthatók ki mind a felhasználó- mind a termékalapú módszerekre, akár a kettő kombinációjára is. Ezek a regressziós modellek tulajdonképpen a szomszéd alapú módszer általánosításának is tekinthetők. Így a kapott súlyok matematikailag pontosabbak, mivel egy optimalizálási formula eredményeként jönnek létre.

2.4. Felhasználó alapú legközelebbi szomszéd regressziós módszer

A már említett normalizált hasonlósági együtthatót helyettesíthetjük egy ismeretlen w_{vu}^{user} változóval, ezzel tudjuk modellezni a becslést az r_{uj} -re:

$$\hat{r}_{uj} = \mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \tag{2.9}$$

Mint a szomszéd alapú modelleknél itt is Pearson korreláció segítségével határozuk meg a $P_u(j)$ halmazt. Azonban a szomszéd modellnél a $P_u(j)$ az u-hoz legközelebbi k darab felhasználót vesszük, akik értékelték j terméket. Így a halmaz mérete általában k nagyságú, ha van k darab felhasználó aki értékelte j tárgyat. A regressziós modellnél először meghatározzuk az u-hoz legközelebbi felhasználókat és ezután csak azokat vesszük figyelembe, akik értékelték a j terméket. Ennek következményeképpen a halmaz sokszor kisebb, mint k. A számítás különbsége és a megfelelő számú adat biztosítása érdekében, a regressziós változatnál a k értékét jóval nagyobbra kell venni.

Az ismeretlen w_{vu}^{user} változó határozza meg az u felhasználó értékeléseire tett becslések egy részét, a befolyásolás mértéke a v-hez való hasonlóságtól függ. w_{vu}^{user} és w_{uv}^{user} különbözhetnek, mivel egy adott felhasználóhoz a Pearson korreláció alapján számolt legközelebbi k felhasználót vesszük figyelembe és ezek különbözhetnek két felhasználó között. A $P_u(j)$ halmazon kívül eső elemek nem fontosak a becslés szempontjából, így ezek csökkentik a számolás idejét. Az alapötlet a már meglévő értékelésekre egy becslés számolása a $P_u(j)$ halmaz segítségével, a valós értékelés és a becsült értékelés különbsége alapján egy optimalizálási modellt állíthatunk fel, ahol a hiba mértékét mérjük. Így egy legkisebb négyzetek optimalizálási problémát hozhatunk létre az ismeretlen w_{vu}^{user} változóra. Az össz értékelt termékre kiszámítjuk a hibák négyzetét, majd ezeket összeadjuk és ez alapján jön létre a legkisebb négyzetek problémája. Az optimalizálási modellt minden u felhasználónak külön állítjuk elő a következőképpen:

$$Minimize \;\; J_u = \sum_{j \in I_u} (r_{uj} - \hat{r}_{uj})^2 = \sum_{j \in I_u} \left(r_{uj} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2 \tag{2.10}$$

A J_u- re kapott értékeket összeadhatjuk, mivel minden egyenletnek megvannak a saját ismeretlen változói, amelyek nem befolyásolják egymást. Így az egész adathalmazt figyelembe véve a következőt kapjuk:

$$Minimize \sum_{u=1}^{m} J_u = \sum_{u=1}^{m} \sum_{j \in I_u} \left(r_{uj} - \left[\mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \right] \right)^2 \tag{2.11}$$

Az értékelések ritkasága miatt probléma keletkezhet, ugyanis $P_u(j)$ mérete jelentősen különbözhet egy adott u felhasználónál különböző j termékekre. A kimagaslóan magas ritkaság szinte az összes értékelési mátrixnál előfordul. Ennek következményeképp a regressziós együtthatókat jelentősen befolyásolják a j terméket értekelő szomszédos felhasználók. Gyakori jelenség a túlilleszkedés (overfitting), mert a statisztikailag megbízhatatlan regressziós együtthatók zajt visznek a becsült értékelésekbe. Erre egy lehetséges megoldás ha a becslésnek csak egy töredékét befolyásolja a j termékre vonatkozó regresszió $\frac{|P_u(j)|}{k}$. Így figyelembe vesszük, hogy a rendelkezésre álló információ hiányos is lehet.

$$\hat{r}_{uj} \cdot \frac{|P_u(j)|}{k} = \mu_u + \sum_{v \in P_u(j)} w_{vu}^{user} \cdot (r_{vj} - \mu_v) \tag{2.12}$$

2.5. Termék alapú legközelebbi szomszéd regressziós módszer

A termék alapú legközelebbi szomszéd regressziós módszer abban különbözik a felhasználó alapútól, hogy a regressziót nem a felhasználók közötti korrelációra alapozzuk, hanem a termékek közöttire. Az u felhasználó t termékre tett értékelés becslésének modellezéséhez egy ismeretlen w_{jt}^{item} változót vezetünk be:

$$\hat{r}_{ut} = \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj} \tag{2.13}$$

$Q_t(u)$ a t-hez leghasonlóbb termékek részhalmaza, amelyeket u felhasználó értékelt, ezeket adjusted cosine-nal számolhatjuk ki. A $Q_t(u)$ mérete jelentősen különbözhet a tradicionális szomszéd alapú módszerben megadotténál, mivel akkor a t-hez legközelebbi olyan termékeket keressük, amelyeket u felhasználó már értékelt. A regressziós módszernél a halmazt a leghasonlóbb termékekből állítjuk össze és itt csak azokat a tárgyakat vesszük figyelembe, amelyeket u felhasználó már értékelt, így a $Q_t(u)$ mérete gyakran sokkal kisebb, mint k. Az adathiány elkerülése miatt ennél a módszernél alapból nagyobb

k értékkel dolgozunk. A w_{jt}^{item} befolyásolja a t termékre tett becsült értékelést, hatása a j termékhez való hasonlóságától függ. Az értékelési mátrix alapján felállíthatunk egy legkisebb négyzetek optimalizálási problémát az ismeretlen w_{jt}^{item} változókra nézve. A probléma célja a meglévő értékelések becslése az adott termékhez leghasonlóbb k terméket felhasználva, majd a két érték különbségének a minimalizálása. Az optimalizálási problémát az összes t cél termékre elvégezzük. Egy t termékre készített minimalizálási feladat a következőképpen néz ki:

$$Minimize \;\; J_t = \sum_{u \in U_t} (r_{ut} - \hat{r}_{ut})^2 = \sum_{u \in U_t} (r_{ut} - \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj})^2 \tag{2.14}$$

Minden termékre külön optimalizálási problémát állítunk elő, ezért összevonhatjuk az összes minimalizálási feladatot és nem befolyásolják egymást, mivel minden egyenlet külön w_{jt}^{item} értékeket tartalmaz. Az egész problémának a formulája:

$$Minimize \sum_{t=1}^{n} \sum_{u \in U_t} (r_{ut} - \sum_{j \in Q_t(u)} w_{jt}^{item} \cdot r_{uj})^2 \tag{2.15}$$

A 2. fejezet a [1, 2] forrás alapján készült.

3. fejezet

Ritkasági lineáris modellek (SLIM – Sparse linear models)

A SLIM módszer a termék alapú regressziós modellnek egy változata, amely a regressziós együtthatók ritkaságára lett kitalálva, mivel ezeket regulizációs metódusokkal egészíti ki. Jelentős különbség az eddigi modellekhez képest, hogy nem negatív értékelési esetekre lett kifejlesztve, ilyenkor a felhasználónak nincs külön lehetősége ”nem tetszést” kifejezni egy adott termékre. Gyakorlatban ez a modell tökéletes az implicit visszajelzési mátrixoknál (pl. klikkeléseket, megvásárlásokat figyelő adathalmazoknál), a rendszer csak a felhasználó általi pozitív cselekedeteket veszi figyelembe. Ilyen modelleknél a hiányzó adatokat (pl. egy adott felhasználó nem vásárolt meg egy terméket) 0-nak vesszük az optimalizálási formula tanításának céljából. Ezekre a 0-ra az optimalizálási modell magas értékű pozitív becslést is tehet és ezek a legjobb ajánlásra alkalmas felhasználó-termék kombinációk. A felhasználó alapú módszerrel ellentétben a SLIM-nél a regressziós együtthatók nem csak a t termék szomszédjaira vonatkoznak. A becslést a következőképpen számoljuk:

$$\hat{r}_{ut} = \sum_{j=1}^{n} w_{jt}^{item} \cdot r_{uj} \qquad \forall u \in \{1 \dots m\}, \forall t \in \{1 \dots n\} \tag{3.1}$$

Az egyenlet jobb feléről ki kell zárnunk a célterméket, ha nem tesszük, akkor ez túlilleszkedéshez (overfitting) vezethet. Ezt úgy érjük el, hogy feltétellé tesszük, hogy $w_{tt}^{item} = = 0$. $\hat{R} = [\hat{r}_{uj}]$ jelöli a becsült értékeket tartalmazó mátrixot és $W^{item} = [w_{tt}^{item}]$ pedig a termék-termék regressziós mátrixot. Az előző feltétel alapján a W mátrix átlójának összes

eleme 0 értéket kap, ekkor az előző egyenletet különböző felhasználók és termék kombinációival alkalmazva a következő mátrix szintű becslést állíthatunk fel:

$$\begin{aligned} \hat{R} &= RW^{item} \\ Diagonal(W^{item}) &= 0 \end{aligned} \tag{3.2}$$

A fő cél a $||R - RW^{item}||^2$ Frobenius norm és a regularizátor kifejezések minimalizálása. W mátrix oszlopai egymástól függetlenek, így az optimalizálási problémát minden t termékre külön oldjuk meg, $w_{tt}^{item} = 0$. Az értelmezhetőség érdekében a regressziós részek összegénél a súly vektoroknak nem negatívnak kell lenniük. Az összes feltétellel együtt a t termékre vonatkozó optimalizálási feladat a következőképpen néz ki:

$$\begin{aligned} Minimize \;\; J_t^s &= \sum_{u=1}^{m}(r_{ut} - \hat{r}_{ut})^2 + \lambda \cdot \sum_{j=1}^{n}(w_{jt}^{item})^2 + \beta \cdot \sum_{j=1}^{n}|w_{jt}^{item}| \\ &= \sum_{u=1}^{m}(r_{ut} - \sum_{j=1}^{n} w_{jt}^{item} \cdot r_{uj})^2 + \lambda \cdot \sum_{j=1}^{n}(w_{jt}^{item})^2 + \beta \cdot \sum_{j=1}^{n}|w_{jt}^{item}| \end{aligned} \tag{3.3}$$

$$w_{jt}^{item} \geq 0 \qquad \forall j \in \{1 \dots n\}$$

$$w_{tt}^{item} = 0$$

A λ és β rész egy elastic-net regularizer, ami kombinálja az $L1$ és $L2$ regulizátorokat. Tulajdonképpen egy Ridge és egy Lasso regresszióból tevődik össze.

3.1. SLIM jelölések

Az u fogja jelölni a felhasználókat, t pedig a termékeket. Egy felhasználó vagy termék esetében az előző jelölést egy index-szel bővítjük ki, pl.: $u_i = i$.-edik felhasználó, $t_j = j$.-edik termék. Az összes felhasználót tartalmazó halmazt U jelöli, amelynek mérete m, vagyis a felhasználók száma. T az összes terméket tartalmazó halmaz, amelynek mérete

n. A felhasználó-termék vásárlási mátrixot A-val jelöljük, amelynek mérete $m \times n$ (felhasználók száma x termékek száma). A mátrix (i, j) -edik eleme, amelyet a_{ij}-vel jelölünk, 0 értékű, ha az u_i felhasználó nem vásárolta meg a t_j terméket, ellenkező esetben 1 lesz az elem értéke. Az A mátrix i.-edik sora az u_i felhasználó vásárlásait tartalmazza a T termék halmazra nézve, ezt a sort a_i^T-vel is jelöljük. A mátrix j.-edik oszlopa pedig az összes felhasználó által tett vásárlásokat tartalmazza a t_j termékre nézve, ezt a_j jelöli. A vektorokat kis betűvel (a_i, w_j stb.), a mátrixokat nagy betűvel (A, W stb.) jelöljük. A sor vektorokat transzponáltként vannak reprezentálva (a_i^T), ellenkező esetben oszlop vektorok (a_i). A becsült értékek felett egy $\wedge$ jelölés található.

3.2. SLIM működése

A SLIM módszernél az u_i felhasználó által még nem megvásárolt t_j termékre a becslést az u_i által eddig vásárolt termékek összesítésével kapjuk, ez az adathalmaz általában ritka. Tehát a_{ij} becsült értékét így számítjuk:

$$\hat{a}_{ij} = a_i^T w_j \tag{3.4}$$

Itt $a_{ij} = 0$ és w_j egy ritka, n hosszú oszlop vektor, amely az összesítési együtthatókból áll. Ha a SLIM módszert az egész A mátrixra nézzük, akkor a következőt adhatjuk meg:

$$\hat{A} = AW \tag{3.5}$$

Itt az A mátrix a felhasználó-termék vásárlási történetet jelöli, amely $m \times n$ méretű. W egy $n \times n$ mátrix, amely az összesítési együtthatókból áll, w_j a W mátrix j.-edik oszlopa. Az $\hat{a}_i^T$ sor vektor az u_i felhasználóhoz tartozó becsléseket tartalmazza az összes termékre ($\hat{a}_i^T = a_i^T W$). A top-N ajánlott terméket u_i felhasználóra úgy kapjuk meg, hogy az összes becsült értéket, amelyet még nem vásárolt meg, csökkenő sorrendbe rendezzük és az első N darabot kínáljuk. Vagyis $\hat{a}_i^T$ rendezzük csökkenő sorrendbe, de csak azokat az x elemeket, amelyekre igaz hogy ai $x = 0$, tehát az u_i még nem vásárolta meg.

3.3. W kiszámolása

Az u_i felhasználó t_j termék vásárlásával kapcsolatos információt (a_{ij}) tekintjük az ajánlás alapigazságának. Ha adott egy $m \times n$ -es A mátrix, akkor a W $n \times n$ -es, ritka mátrixot a következő minimalizálási problémával kaphatjuk meg:

$$minimize\ w \quad \frac{1}{2}||A - AW||_F^2 + \frac{\beta}{2}||W||_F^2 + \lambda||W||_1 \tag{3.6}$$

$$W \geq 0$$

$$diag(W) = 0$$

$||W||_1 = \sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}|$ jelöli a $W1$ L1-normálját és $||\cdot||_F$ pedig a W mátrix Frobenius normálját.

Az egyenlet első része a négyzetek különbsége, azt vizsgálja, hogy milyen jól illeszkedik a lineáris modell a tanuló adatra. A másik két rész, a normálok pedig regulizátorokként funkcionálnak. A λ és β konstansok regulizációs paraméterek. Minél nagyobb értékűek, annál jobban befolyásolják az egyenletet. A W mátrixra vonatkozik egy nem negativitási feltétel, azért, hogy a termékek között, ha van összefüggés, akkor pozitív legyen, a negatívat nem nézzük. A W mátrix másik feltétele, hogy az átlója 0-ból áll, ez azért szükséges, mert ellenkező esetben előfordulhat, hogy egy adott termék önmagát ajánlja, mert ez ”hasonlít rá legjobban”. Így a_{ij} nem befolyásolja az $\hat{a}_{ij}$ -re számolt becslést.

Az W $L1$-norm regulizátor ritkaságot vezet be a megoldásba, míg az Lf-norm a modell komplexitását méri és megakadályozza az túlilleszkedést. Együttesen egy elasztikus háló problémát alkotnak, amely egy ridge és egy lasso regresszióból tevődik össze.

Mivel a W mátrix oszlopai függetlenek, ezért az előző nagy mátrix szintű minimalizálási problémát szétbonthatjuk oszloponként, így a következő egyenletet kapjuk:

$$minimize\ \ w_j \quad \frac{1}{2}||a_j - Aw_j||_2^2 + \frac{\beta}{2}||w_j||_2^2 + \lambda||w_j||_1 \tag{3.7}$$

$$w_j \geq 0$$

$$diag(w_{jj}) = 0$$

Ha az összes j-re $(1 \ldots n)$ kiszámoljuk, akkor megkapjuk W-t.

3.4. SLIM ”feature selection”

Gyakran óriási méretű adathalmaz alapján kell becslést számolni, ez jelentősen megnöveli a számolási igényt ha az egész mátrixot figyeljük. Az oszlopok számának növekedésével egyre nő a futási idő és a jelentéktelen adat is, mivel teljesen eltérő termékeket is figyelembe veszünk. A probléma elkerülése érdekében csökkenthetjük a becslés felhasznált oszlopok számát. Az A mátrix a_j-hez leghasonlóbb k oszlopaira szűrhetjük a releváns adathalmazt. Az oszlopok hasonlóságának a kiszámításának egyik módszere ”adjusted cosine similarity”.

A 3. fejezet a [1, 3] forrás alapján készült.

4. fejezet

Megvalósításhoz használt technológiák

A SLIM algoritmust az Eclipse JEE fejlesztői környezetben valósítottam meg, Java 8. verziót használtam. Az ajánlórendszert egy webes környezetbe is implementáltam, aminek backend része egy Spring boot-os megvalósítás, míg a frontend-en Thymeleaf templateket használtam, a szükséges .jar fájlokat a Maven nevű build automatizálóval töltöttem be. Az adatok tárolására a MySQL adatbázist választottam. A fejlesztés folyamán a frissítéseket a Github verziókövető rendszerbe is feltöltöttem.

4.1. Java

A Java [4] egy általános célú, objektumorientált programozási nyelv, melyet a 1995-ben adott ki a Sun Microsystems és egészen 2011-ig fejlesztették is, amikor az Oracle felvásárolta a céget. A Java nyelven megírt alkalmazásokat általában bájtkódra alakítjuk, amelynek futtatását a Java virtuális gép (Java Virtual Machine, JVM) végzi. Szintaxisa a C és a C++ nyelvekéhez hasonló, viszont kevesebb alacsony szintű művelet áll rendelkezésre. Nagy előnye a C++ nyelvvel szemben, hogy könnyebben elsajátítható.

4.2. Eclipse

Az Eclipse [5] egy Java nyelven megírt, nyílt forráskódú, integrált fejlesztői környezet (Integrated Development Environment, IDE), amely a programozáshoz használható segítőeszköz. Elsősorban Java nyelvű alkalmazások fejlesztésére használják, de egyéb nyelve-

ket is támogat. Az IDE részét képezik a szövegszerkesztő, fordító és a debugger, amelyek grafikus felületen keresztül használhatóak. Funkcionalitása tovább bővíthető pluginekkel.

4.3. Spring Boot

A Spring Boot [6] egy nyíltforráskódú, Java nyelven írt keretrendszer, amely a Spring keretrendszerre épül, a Pivotal cég nevéhez fűződik. Egy felületet ad a fejlesztőknek, amellyel automatikusan konfigurált Spring applikációt tudnak készíteni. Minimális konfigurálás után futtatható applikációt készíthetünk, amelyet egy beépített Tomcat szerveren tudunk üzemeltetni. Nagy előnye, hogy jelentősen csökkenti a fejlesztési időt, mivel sok beállítást automatikusan elvégez.

4.4. Thymeleaf

A Thymeleaf [7] egy server oldali Java template motor, amelyet web és egyedülálló alkalmazásoknál egyaránt használnak. HTML és CSS templatek integrálását segíti elő.

4.5. Maven

Az Apacha Maven [8] egy build automatizálásra használt eszköz, amelyet elsősorban Java projekteknél alkalmaznak. Szorosan kötődik hozzá a POM fogalma, vagy Projekt Objektummodell (Project Object Model), amely a build-elendő projektet és a hozzá tartozó függőségeket írja le, ezt egy XML fájlon keresztül teszi. A Maven az XML fájlban definiált célokat elvégzi, mint például: a kód lefordítása, megfelelő Java könyvtárak letöltése, csomagolás. A letöltéseket egy központi repository-ból tölti le és egy lokális cache-ben menti el.

4.6. MySQL

A MySQL [9] egy nyílt forráskódú relációs adatbázis-kezelő rendszer (relational database management system, RDBMS). Eredetileg a MySQL AB cég tulajdona volt, amelyet

később megvásárolt az Oracle.

4.7. Git

A Git [10] egy nyílt forráskódú, verziókezelő szoftver (version control system, VCS), amely Linus Torvalds nevéhez kötődik. Lehetővé teszi a forráskód verzióinak követhetőségét, amely a fejlesztés folyamán folyamatosan bővül. Elsősorban csapatszintű projektek támogatására szolgál, de egyéni projekteknél is előnyt jelenthet. A forráskódot és az egyéb mappában tárolt fájlokat a helyi gépünkön egy lokális repositoryaban tárolhatjuk, amelyet csak mi érünk el, ennek akár több verziója is lehet, amelyet különböző ágakon (branch) tárolunk. A fájlainkat egy távoli (remote) repositoryba is feltölthetjük, amely mások számára is láthatóvá teszi a kódot.

5. fejezet

A SLIM algoritmus megvalósítása

Az egész ajánlórendszert képező egység a com.szte.recommender.project.recommender package-ben található, amely 7 osztályt tartalmaz:

- SimilarityCalculator.java
- DerivationHelper.java
- ExpressionHelper.java
- GaussEliminationHelper.java
- Recommender.java
- RecommenderHelper.java
- VectorHelper.java

5.1. SimilarityCalculator

A SimilarityCalculator osztály feladata a termékek közötti hasonlóság mértékének a meghatározása.

- meanRating - Két paraméterrel rendelkezik, az első a matrix, amely az A mátrixot jelöli, a második pedig a céltermék oszlopának az indexét tartalmazza. A függvény kiszámolja a céltermék oszlopának az átlag értékelését.

- meanCentered - A paraméterben kapott értékek alapján a matrix[row][itemIndex] értékből kivonja a termék átlag értékelését, így megkapjuk a súlyozott átlagot.

- adjustedCosine - Két termék hasonlóságát méri. A termékek oszlopait az itemIndex1 és itemIndex2 jelöli. A hasonlóság értéke egy [-1, 1] intervallumba tartozó érték, minél nagyobb az érték, annál jobban hasonlít a két termék.

- similarityOfItems - A targetItem paramétere a céltermék, amelyhez hasonlítja az összes többi terméket, ehhez az adjustedCosine() függvényt hívja segítségül.

5.2. DerivationHelper

A DerivationHelper osztály egy külső .jar fájl (javacalculus.core.CALC) segítségével végzi el a minimalizáláshoz szükséges deriválással kapcsolatos számolásokat.

- getCalcObjects – Az expression nevű paraméter egy deriválási egyenlet lesz, a vectorW paraméter pedig az egyenletben található összes ismeretlen változót tartalmazó tömb. A függvény az egyenletet összes ismeretlen változója szerint egyenként derivál. Egy derivált egyenletet a results nevű tömbben tárol el, amelynek típusa CalcObject. A deriváláshoz a getDerivative függvényt hívja meg.

- getDerivative – Az expression paraméter maga az egyenlet, a variable argumentum pedig az ismeretlen változó, amely szerint az egyenletet deriválja. A deriválás számolását a CALC osztály végzi, amelyet külön importáltunk. Egy CalcObject nevű objektumot ad vissza, amely a variable szerint derivált egyenletet tárolja el.

- decodeDerivative – A calc paraméter a derivált egyenletet tartalmazza, a vectorW pedig az egyenletben található ismeretlen változókat. A derivált egyenletet egy Stringként kezeli és szétszedi konstans értékek szerint, mert ezeket akarja kinyerni. Az értékeket 0.5-tel szorozza, mert a SLIM algoritmusban az egész egyenlet ennyivel kerül beszorzásra. Egy double tömbbel tér vissza, amely a derivált egyenletben található konstans értékeket tartalmazza, ezek által állít elő a rendszer egy Gauss eliminálási mátrixot.

5.3. ExpressionHelper

Az ExpressionHelper osztály feladata a deriváláshoz szükséges egyenlet előkészítése String típusként.

- generateWholeExpression – A vector paraméter a SLIM-ben definiált $a_j - Aw_j$ részt tartalmazó vektor, a lambda és beta paraméter a regulizátor paraméterek, a vectorW az ismeretlen változókat tartalmazó tömb. A függvény megadott paraméterek alapján összeállítja az egész egyenletet, amely eleget tesz a deriváláshoz használt eszköznek.
- squareSingleExpression – Az input paraméter egy String érték, amely egy kifejezést jelöl. A függvény feladata, hogy a String paramétert négyzetre emelje.
- getRidgeRegressionPart – A lambda egy regulizátor paraméter, míg a vectorW az ismeretlen változókat tartalmazó tömb. Egy String-et készít, amely a vectorW tömb összes értékének a négyzetét megszorozza a lambda értékkel.
- getLassoRegressionPart – A beta egy regulizátor paraméter, a vectorW az ismeretlen változókat eltároló tömb. Egy String-et készít, amely a vectorW tömb összes értékének az abszolút értékét megszorozza a beta értékkel.

5.4. GaussEliminationHelper

A GaussEliminationHelper osztály feladata a Gauss eliminálás elvégzése és az ehhez szükséges adatok megfelelő formátumba való alakítása.

- getAforGauss – Egy mátrixot kap paraméterként, amit leszűr csak a bal részre. A mátrix utolsó oszlopát nem veszi figyelembe, mivel ez a deriválási egyenletek jobb oldalát képezik.
- getBforGauss – Egy mátrixot kap paraméterként, amit leszűr csak az utolsó oszlopra, ugyanis ez képezi deriválási egyenletek jobb oldalát.

– generateGaussMatrix – A calcObjects a derivált egyenletek tömbje, ez alapján készül a mátrix, ehhez a DerivationHelper osztály decodeDerivative metódusát hívja segítségül.

– calculateGauss – Gauss eliminálást végez a mátrixon, aminek bal részét az A mátrix képvisel, jobb részét pedig a b vektor.

5.5. RecommenderHelper

A RecommenderHelper osztálynak feladata egy adott elemre a becslés számolása és a becsült értékek közül a legjobbak kiválasztása.

– predictRating – A matrix paraméter az egész vásárlási mátrixot képviseli, a row a felhasználó indexe lesz, ez alapján tudjuk, hogy a mátrix melyik sorát kell figyelnünk, a column paraméter a mátrix egy oszlopát képviseli. A függvény a matrix[row][column] elemre számol egy becslést.

– getBestKIndices – Egy array nevű paramétert fogad, amiből szűri ki a legjobb num darabot és ezeknek az indexeivel tér vissza. Egy belső IndexValuePair osztályt használ, ami az index és az érték adattagokat tartalmazza.

– generateNewMatrix - A leghasonlóbb termékek alapján egy csökkentett méretű mátrixot készít, így a nem releváns adatokat kiszűrjük, amely csökkenti a számítási igényt és növeli a pontosságot is.

5.6. VectorHelper

A VectorHelper osztály feladata a szükséges vektor műveletek elvégzése.

– generateVariableVector – Két paramétere van, az első a size, amely azt adja meg, hogy hány ismeretlen változó szükséges, a columnIndex argumentum pedig a számoláshoz szükséges mátrix oszlopot indexét jelöli. A függvény feladata ismeretlen változók előállítása (SLIM-ben a w_j vektor), ezt úgy teszi, hogy String-eket készít. A String a következőképpen épül fel: w (ez jelöli, hogy vektor) [0 … size) (a sor

indexét fogja jelölni) x (csak egy elválasztó karakter a sor és az oszlop index között) columnIndex (az oszlop indexét fogja jelölni).

– multiplySameSizeVectors – Két azonos méretű vektort kap paraméterként és ezeket összeszorozza.

– matrixTimesWvector – Az első paramétere a matrix, amely az A mátrixot jelöli, a második argumentum pedig az ismeretlen változókat tartalmazó tömb. A függvény feladata, hogy a két paramétert összeszorozza és egy String formában adja vissza.

– columnMinusVector – A vectorA paraméter az A mátrix egy oszlopát jelöli, míg a A_vectorW a matrixTimesWvector függvény eredményét tartalmazza. A függvény feladata, hogy kijelölt vectorA oszlopból kivonja a A_vectorW vektort és ezt egy String típusként adja vissza.

– getColumn – Az első paraméter egy mátrix, a második pedig egy oszlopnak az indexe. A függvény pedig a kapott mátrix indexedik oszlopát adja vissza.

5.7. Recommender

A Recommender osztály köti össze az ajánlórendszert az applikáció Service rétegével. A Service rétegtől kapja az adatokat, amelyekkel tud a bejelentkezett felhasználónak egy ajánlást számolni.

– initMatrix – Az első paraméter az adatbázisból kapott felhasználók listája, a második pedig a könyvek listája. A függvény feladata a felhasználók vásárlási története alapján az A mátrix elkészítése.

– contains – A purchases paraméter egy felhasználó vásárlásainak az azonosítóját tartalmazza String formátumban, az id paraméter pedig egy könyv azonosítóját jelöli. A függvény eldönti, hogy a felhasználó megvásárolta-e az adott könyvet.

– getPredictedRatings – Az első paraméter a könyvek számát jelöli (vagyis n értékét), míg a második userId paraméter a bejelentkezett felhasználó azonosítóját tartalmazza. A függvény feladata az adott felhasználó által még nem megvásárolt termékekre

egy becslés számolása, majd a becsült értékek alapján a legjobb 10 könyv indexével tér vissza.

– getUserIndexInList – A bejelentkezett felhasználó azonosítója az első paraméter, a második pedig a felhasználók listája. A függvény megkeresi, hogy a listában hányadik helyen található az adott felhasználó.

– recommendedBooksForUser – Az adatbázisból kapott felhasználók és könyvek listáját kapja paraméterként. Az előző függvényeket felhasználva állítja össze az ajánlást és a getPredictedRatings által visszaadott indexek segítségével ezeknek a könyveknek a listájával tér vissza.

5.8. Program felépítése

Az adatok tárolásáért a MySQL adatbázis felelős. Az adatbázis eléréséhez a mysql driver szükséges. A kapcsolathoz szükséges adatok a src/main/resource mappában található application.properties fájlban vannak megadva. A repository rétegen keresztül tudunk lekérdezéseket küldeni az adatbázisnak. A service rétegben megfelelő repository objektumokat hozunk létre, ezeken keresztül lekérjük az adatot, és itt végezzük el a szükséges backend számításokat. A controller rétegben található osztályok a megfelelő service osztályokat példányosítják és ezeken keresztül megkapjuk a számolásokon átesett adatokat. A @GetMapping annotációkban megadott URL-eket figyelik az alattuk lévő metódusok, amikor a böngésző meghívja az adott URL-t, akkor a kontroller visszaad egy thymeleaf template-t. A templatek a src/main/resources/static mappában találhatóak, ezek .html fájlok, amelyek a metódusban átadott Model alapján elérik a Java objektumokat és így dinamikusan beilleszthetők. A felhasználó ezeket a .html fájlokat látja amikor böngészőn megnyitja az oldalt.

5.9. Tesztelés

A teszteléshez egy 30×30 tetszőleges adatokat tartalmazó mátrix készült. A felhasználók különböző érdeklődési körrel rendelkeznek és eddigi vásárlásaik ezt reprezentálják. A

könyveket 7 műfajra oszthatjuk (sci-fi, egészség, történelmi, fantasy, szerelmi, önéletrajz, pszichológia). Az adathalmaz kép formátumban (5.1 és 5.2 ábrák) a mellékletekben és elektronikus mellékletekben lévő ”ajanlorendszer_adatok” fájlban található. A teszteknél 4 paraméter befolyásolja az eredményeket, ezek: lambda, beta, bejelentkezett felhasználó, leghasonlóbb termékek halmazának a mérete (továbbiakban: lthm). A lambda és beta paraméterek a becslés pontosságára hatnak, míg a lthm mind a pontosságot, mind a futási időt jelentősen befolyálja. A teszteléshez használt processzor: i5 - 6300hq , rendelkezésre álló belső memória: 8 GB.

Lambda	Beta	Felhasználó ID	Lthm	Futási idő	Top 5 könyv ID
1.7	0	8	6	2557 ms	4, 25, 28, 22, 29
1.7	1.4	8	6	2187 ms	25, 4, 28, 29, 0
0.5	1.4	8	6	2063 ms	25, 29, 0, 1, 2
0.5	1.4	8	4	1145 ms	29, 25, 0, 1, 2
0.5	1.4	15	4	538 ms	29, 18, 19, 0, 1
2.3	1.4	15	4	1375 ms	18, 19, 23, 24, 29
2.3	0.3	15	4	1558 ms	23, 24, 19, 18, 29
2.3	0.3	15	9	6213 ms	23, 24, 19, 18, 29
2.3	0.3	21	9	6556 ms	28, 27, 4, 25, 20
2.3	0.3	21	7	3294 ms	28, 27, 4, 25, 20
2.3	0.3	21	5	1652 ms	28, 4, 20, 27, 16
2.3	1.7	21	5	1563 ms	4, 28, 10, 0, 1
5.6	1.7	21	5	1769 ms	28, 4, 27, 10, 20
5.6	1.7	24	5	944 ms	15, 0, 10, 6, 7
5.6	1.7	24	10	8948 ms	15, 21, 10, 7, 20
3.8	1.7	24	10	8746 ms	15, 21, 7, 20, 22
3.8	0.2	24	10	9386 ms	15, 3, 0, 10, 11
3.8	0.2	24	17	71390 ms	15, 0, 3, 10, 21

A lambda és beta paraméterek a futási időre nem voltak hatással, a becsült értékeken pontosítottak, azonban sok esetben ez a legjobb becsléssel rendelkező 5 termék sorrendjét csak kicsit befolyásolta. Az lthm növelése jelentősen megnöveli a futási időt és ez nem feltétlenül javít a becslés pontosságán. Túl nagy szomszéd halmaz esetén olyan adatokat is figyelembe veszünk, amelyek nem segítik elő a becslés pontosságát. A teszthalmazon a legjobb lhtm értékek az $[5 - 7]$ intervallumba tehetők, ekkor kapunk egy megfelelő méretű szomszéd halmazt, amely alapján pontosabb becsléseket adhatók és a futási idő is elfogadható.

Összegzés

A szakdolgozattal a következő eredmények valósultak meg:

1. Egy pozitív értékelési módszerre alapuló explicit ajánlórendszer megtervezése
2. A SLIM algoritmus Java alapú megvalósítása
3. A futási idő csökkentésének érdekében az úgynevezett ”feature selection” bevezetése
4. A becslés pontosságának szabályozásához ”ridge” és ”lasso” regresszió megvalósítása
5. Az ajánlórendszer implementálása webes környezetbe

A SLIM egy hatékony explicit ajánlórendszer megvalósítását teszi lehetővé, amely problémához mérten konfigurálható. A ”ridge” és ”lasso” regressziós paraméterek lehetővé teszik a becslések pontosságának finomhangolását. A leghasonlóbb termékek halmazának a mérete is állítható, amely a legnagyobb változást okozhatja, a futási időn és a becslés pontosságán is javít. A minta 30×30 mátrix esetén az [5-7] intervallumba tartozó értékek voltak a legoptimálisabbak. Kisebb értéknél a számolás gyors, viszont a pontosság mértéke jelentősen csökken, mivel nincs elég adatunk, ezért nehéz becslést adni. Nagyobb értéknél a számolás lassú és a pontosság sem biztosan javul, mivel túlságosan sok adat alapján tesz becslést a rendszer, így sok felesleges információ is befolyásolja a számolást.

Irodalomjegyzék

[1] Charu C. Aggarwal. *Recommender Systems*, pp. 1-60

[2] Michael D. Ekstrand, John T. Riedl és Joseph A. Konstan. *Collaborative Filtering Recommender Systems*, pp. 82-107

[3] Xia Ning és George Karypis. *SLIM: Sparse Linear Methods for Top-N Recommender Systems. IEEE International Conference on Data Mining*, pp. 497-506, 2011.

[4] Java **https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf**

[5] Eclipse **https://help.eclipse.org/2019-09/index.jsp**

[6] Spring Boot **https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/**

[7] Thymeleaf **https://www.thymeleaf.org/documentation.html**

[8] Maven **https://maven.apache.org/guides/**

[9] MySQL **https://dev.mysql.com/doc/**

[10] Git **https://opensource.com/resources/what-is-git**

Mellékletek

A táblázat túl nagy, ezért csak feldarabolva (2 részre) fér el.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	A Perdide árvája	Praotorianus	Mulandó üresség	Dűne	Önerőből	Keto	Újra én	Élni jöttem	A nagy történelem	Micsoda népek	Alkony	Trianon	Vaják - Tündevér	Hobbit	A kiválasztott	Eragon - Örökség
user0		1	1	1												
user1					1			1								
user2									1	1						
user3														1		1
user4																
user5																
user6																
user7	1	1												1	1	
user8						1	1	1								
user9										1		1			1	1
user10									1	1	1					
user11	1			1									1		1	
user12									1			1	1	1		1
user13																
user14																
user15																
user16											1	1				
user17																
user18					1		1									
user19																
user20																
user21						1	1									
user22																
user23										1	1					
user24		1	1						1	1			1	1	1	
user25					1	1		1			1					
user26														1		1
user27					1			1								
user28					1	1	1						1		1	1
user29											1	1				

5.1. ábra. Táblázat bal része (0 - 15 id-val rendelkező könyvek)

16	17	18	19	20	21	22	23	24	25	26	27	28	29	
Most élsz	A stáb	Maradj közel	Hallgass meg!	Michael Jordan - A levegő Ura	Ferkovics József	Élet	Hivatásom prizmája	Visszadobtak	A nő	A férfi	A kiút	A képlet	A fájdalom arcai	
														user0
														user1
														user2
														user3
1	1		1											user4
				1		1	1							user5
									1		1	1	1	user6
														user7
										1	1			user8
														user9
					1		1							user10
														user11
														user12
1		1								1	1		1	user13
	1	1	1		1		1							user14
1	1			1	1	1								user15
				1			1							user16
						1	1	1	1			1		user17
										1	1	1		user18
				1	1				1			1	1	user19
	1		1						1	1	1			user20
						1		1					1	user21
1	1				1		1			1	1	1	1	user22
1		1	1	1	1			1						user23
														user24
									1		1			user25
1	1	1												user26
				1	1		1	1						user27
														user28
						1		1	1	1	1	1		user29

5.2. ábra. Táblázat jobb része (16 - 29 id-val rendelkező könyvek)

Nyilatkozat

Alulírott Dodony Róbert programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Számítógépes Algoritmusok és Mesterséges Intelligencia Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Dátum: 2019.12.06 Aláírás:

Köszönetnyilvánítás

Szeretném megköszönni témavezetőmnek, dr. Dombi Józsefnek a szakdolgozatom elkészítéséhez nyújtott segítséget. Továbbá szeretném megköszönni Tekeli Tamásnak a SLIM algoritmus matematikai részében nyújtott segítségét. Köszönöm anyámnak, apámnak, testvéremnek és Szkocsovszki Zsoltnak, hogy erőt adtak és támogattak.