UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

Proiect de diplomă

APLICAȚIE PENTRU EVALUAREA MODELELOR DE RECOMANDARE

Absolvent
Diyar Mert Daniel

Coordonator științific
Conf. dr. Andrei Pătrașcu

București, septembrie 2023

Rezumat

Evaluarea și compararea modelelor de recomandare reprezintă un aspect crucial în dezvoltarea și implementarea sistemelor de recomandare eficiente. Există o varietate de algoritmi de recomandare, astfel că evaluarea acestora permite identificarea celui mai potrivit model pentru domeniul sau scopul specific.

Aplicația web implementată oferă utilizatorului o interfață pentru compararea mai multor modele de recomandare, prin selectarea a diferitor modele uzuale, ce mai apoi sunt evaluate conform unor metrici din documentația deja existentă în domeniu cât și folosind două metrici cu implementare proprie, iar mai apoi comparând rezultatele modelelor între ele.

Tehnologia folosită pentru implementarea aplicației este Python, utilizată atât pentru antrenarea și evaluarea modelelor, definirea metricilor de evaluare și a altor funcții utilitare, cât și pentru realizarea aplicației web, alături de JavaScript, ce facilitează tratarea diferitor acțiuni la nivelul interfeței aplicației.

Abstract

Evaluating and comparing recommendation models is a crucial aspect in the development and implementation of effective recommendation systems. There is a variety of recommendation algorithms, thus considering evaluating them allows the identification of the most suitable model for the specific domain or purpose.

The implemented web application provides the user with an interface to compare multiple recommendation models. The evaluation is performed using well-defined evaluation metrics from the existing documentation for recommendation systems, as well as two metrics that measure robustness and prediction accuracy in situations of data scarcity.

The technology used to implement the application is Python, employed for both training and evaluating models, defining evaluation metrics and other utility functions, as well as creating the web application, alongside JavaScript, a scripting language which facilitates handling various actions at the application's interface level.

Cuprins

Listă de figuri

Capitolul 1

Introducere

1.1 Nevoia pe care o atinge aplicația

O aplicație de evaluare a modelelor de recomandare abordează problema centrală a evaluării și îmbunătățirii performanței sistemelor de recomandare.

Aceste sisteme trebuie să prezică cu precizie și să ofere recomandări care se aliniază cu preferințele utilizatorilor. Din moment ce diverse modele posedă avantaje și limitări distincte, aplicația joacă un rol esențial în analizarea și contrastarea acestor diferențe.

1.2 Soluția propusă

Aplicația implementată reprezintă o soluție de vizualizare a rezultatelor obținute de diverse modele de recomandare uzuale, rezultate asupra cărora au fost aplicate următoarele metrici din documentația deja existentă în domeniu [8] [9]:

- eroarea medie absolută
- rădăcina erorii mediei pătratice
- precizie
- evocare
- scorul F1

În afara acestora, am integrat și două metrici cu implementare proprie pentru a observa rezultatele modelelor în situația deficitului de date sau în situația în care utilizatorii ce nu dau dovadă de preferințe alterează rezultatele generale ale modelelor.

Predicțiile obținute în urma evaluării sunt afișate în formatul unui grafic logaritmic pentru a putea observa cu ușurință diferențele dintre modelele alese.

Aplicația utilizează două seturi de date pentru antrenarea modelelor, anume MovieLens 100K si MovieLens 1M [5].

1.3 Motivul alegerii

Am ales aceasta temă din dorința de a observa modul de funcționare al sistemelor de recomandare și de a crea un proiect prin care să pot vizualiza cu ușurință capacitatea modelelor de recomandare de a înțelege preferințele utilizatorilor dintr-un set de date. Un alt obiectiv a fost dorința de a realiza și o interfață prin care sa putem schimba parametrii metricilor de evaluare, pentru a observa rezultatele acestor modele în diferite situații.

Capitolul 2

Sisteme de recomandare

2.1 Ce sunt sistemele de recomandare

Conform definiției din [13], sistemele de recomandare sunt programe ce au ca scop recomandarea utilizatorilor săi a celor mai potrivite produse sau servicii. Acestea se bazează pe estimarea interesului unui utilizator față de un anumit produs, luând în considerare informații relevante atât despre produsele respective cât și despre utilizatori și interacțiunile cu acestea. Obiectivul dezvoltării sistemelor de recomandare este de a reduce supraîncărcarea informațională prin selectarea celor mai relevante informații dintr-un volum mare de date, oferind astfel opțiuni personalizate. Caracteristica cea mai importantă a unui sistem de recomandare constă în abilitatea sa de a estima preferințele și interesele unui utilizator prin analizarea comportamentului acestuia și/sau al altor utilizatori, generând astfel recomandări personalizate.

2.2 Cum funcționează un sistem de recomandare

Modul de funcționare al unui sistem de recomandare este de realizat prin procesul de colectare și procesare a datelor despre utilizatorii si produsele vizate. Acesta folosește diverse tehnici și algoritmi pentru a prezice interesul unui utilizator pentru anumite produse și pentru a genera recomandări personalizate.

Procesul general al unui sistem de recomandare implică următorii pași:

1. **Colectarea datelor**: Informațiile despre utilizatori, elemente recomandate și interacțiunile dintre acestea sunt colectate și stocate într-o bază de date.

2. **Preprocesarea datelor**: Datele colectate sunt triate, transformate și pregătite pentru procesare. Acest pas poate include eliminarea datelor lipsă, normalizarea valorilor sau reducerea dimensiunii setului de date.

3. **Analiza datelor**: Sistemele de recomandare utilizează diferite tehnici și algoritmi pentru a extrage informații relevante din setul de date. Acestea pot include algoritmi de filtrare colaborativă, filtrare bazată pe conținut, învățare automată sau tehnici de inteligență artificială.

4. **Generarea recomandărilor**: Pe baza analizei datelor, sistemul de recomandare generează recomandări personalizate pentru fiecare utilizator. Acest lucru se realizează prin potrivirea intereselor și preferințelor utilizatorului cu elementele recomandate ce sunt relevante.

2.3 Metode de recomandare

Există mai multe metode de recomandare, dintre care cele mai comune sunt:

1. **Filtrarea bazată pe conținut**: Această metodă este una dintre cele mai simple metode din cadrul sistemelor de recomandare și a fost folosit în special în etapele inițiale ale dezvoltării acestor sisteme. În acestă metodp, se analizează caracteristicile și atributile elementelor pentru a le asocia cu preferințele utilizatorilor. De exemplu, dacă un utilizator a apreciat mai multe filme de acțiune în trecut, sistemul de recomandare bazat pe conținut va sugera alte filme de acțiune similare. [10]

   Această metodă se concentrează pe caracteristicile intrinseci ale elementelor și nu are în vedere interacțiunile sociale sau preferințele altor utilizatori.

2. **Filtrarea colaborativă**: Această metodă se bazează pe colectarea și analiza datelor referitoare la preferințele utilizatorilor și la interacțiunile acestora cu elementele pe care le-au evaluat. Algoritmul identifică utilizatori cu preferințe similare și recomanda elemente pe baza preferințelor altor utilizatori cu interese similare. Există două tipuri principale de filtrare colaborativă: filtrarea colaborativă bazată pe utilizatori și filtrarea colaborativă bazată pe elemente.[10]

3. **Filtrarea bazată pe popularitate**: Această metodă recomandă elementele populare și apreciate de un număr mare de utilizatori. Algoritmul se bazează pe ratingurile și recenziile elemenftelor pentru a determina popularitatea lor.

4. **Filtrarea hibridă**: În această metodă sunt combinate mai multe metode de recomandare pentru a obține rezultate mai precise și personalizate. Pentru a face recomandări, algoritmul poate utiliza atât informații despre preferințele utilizatorilor, cât și despre caracteristicile elementelor. [10]

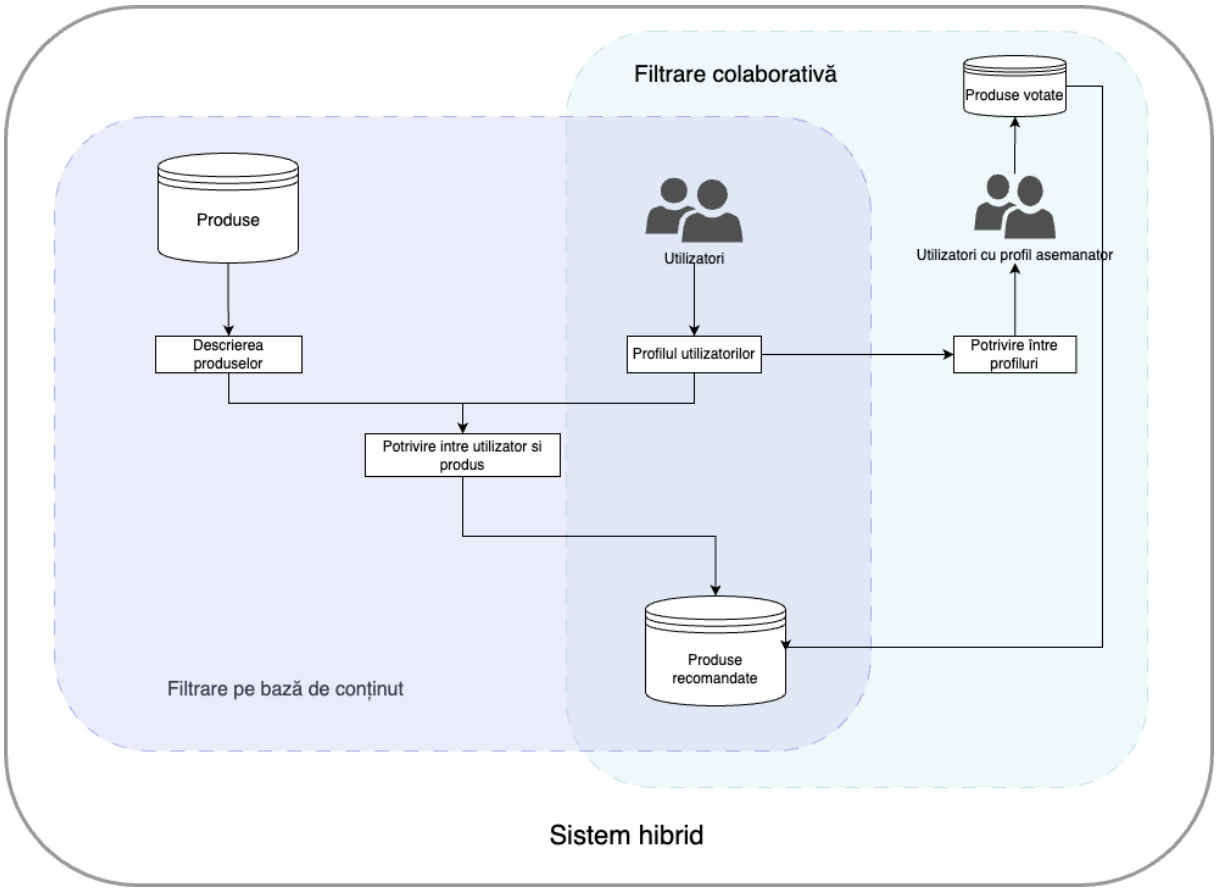

Figura 2.1: Diagrama metodelor de recomandare
[10]

2.4 Tipuri de modele de recomandare

În continuare sunt prezentate următoarele modele de recomandare:

- Baseline
- K-Nearest Neighbours
- Slope One
- De grupare (Co-Clustering)
- Descompunere in valori singulare (SVD)

2.4.1 Modelul Baseline

Modelul Baseline [11] folosește valorile de bază ale elementelor sau ale utilizatorilor și implementează o metodă simplă și eficientă de recomandare.

Modul de funcționare al acestui model este descris în cadrul următorilor pași:

1. **Găsirea valorilor de bază**: Modelul pornește de la premisa că evaluările utilizatorilor pot fi influențate de factori de bază, cum ar fi tendința generală a acestora de a evalua mai sus sau mai jos decât media globală. Astfel, modelul estimează două valori de bază: una pentru fiecare utilizator și una pentru fiecare element (produs, film, cărți etc.).

2. **Estimarea evaluărilor**: Modelul utilizează estimările valorilor de bază și calculează estimări ale evaluărilor utilizatorilor și ale elementelor.

3. **Ajustarea valorilor de bază**: Pentru a obține estimări mai precise, modelul ajustează valorile de bază prin potrivirea modelelor la datele de antrenament. Acest proces implică găsirea valorilor de bază care minimizează eroarea dintre estimările modelului și evaluările reale din setul de antrenament.

4. **Generarea recomandărilor**: Pe baza estimărilor obținute, modelul poate genera recomandări pentru utilizatori. De exemplu, pentru un utilizator care nu a evaluat un anumit element, modelul poate sugera un rating estimat pe baza valorii de bază asociate utilizatorului și elementului respectiv.

2.4.2 Modelul K-Nearest Neighbours

Modelul K-Nearest Neighbours, în rom. *cei mai apropiați K vecini*, se concentrează pe similaritățile dintre utilizatori sau elemente pentru a genera sugestii relevante.

Modul de funcționare al acestui model este descris în cadrul următorilor pași:

1. **Colectarea datelor**: Se adună informații despre preferințele utilizatorilor și evaluările pe care aceștia le-au acordat elementelor. Aceste evaluări pot fi, de exemplu, scoruri, ratinguri sau preferințe exprimate.

2. **Calcularea similarității**: Se calculează gradul de similaritate între utilizatori sau elemente folosind măsurători precum coeficientul de similaritate cosinus sau, în cazul modelului folosit în această lucrare, diferența medie pătratică.

   Formula de calcul a diferenței mediei pătratice este următoarea:

$$MSD(A, B) = \frac{1}{n} \sum_{i=1}^{n} (rating_A - rating_B)^2 \tag{2.1}$$

   unde n reprezintă numărul de elemente comune evaluate de ambii utilizatori, iar suma se calculează pentru toate elementele comune.

3. **Selectarea vecinilor**: Pe baza similarităților calculate, se identifică cei mai asemănători utilizatori (vecini). Numărul de vecini selectați este definit de parametrul K din KNN.

4. **Generarea recomandărilor**: Se determină recomandările prin luarea în considerare a preferințelor utilizatorilor vecini. De exemplu, într-un sistem de recomandare pentru filme, se pot sugera utilizatorului curent filme apreciate de vecinii săi care au rating-uri similare.

5. **Evaluarea și ajustarea modelului**: În cazul în care performanța este sub așteptări, se pot ajusta parametrii, precum numărul de vecini (K), sau se poate îmbunătăți algoritmul de calcul al similarității.

2.4.3 Slope One

Modelul Slope One[12] se bazează pe principiul că preferințele unui utilizator în ceea ce privește elementele pot fi estimate prin analiza diferențelor de evaluare a mai multor utilizatori pentru aceleași elemente.

Modul de funcționare al acestui model este descris în cadrul următorilor pași:

1. **Colectarea datelor**: Modelul începe prin colectarea evaluărilor utilizatorilor pentru diferite elemente.

2. **Calculul deviațiilor**: Modelul calculează deviațiile între perechile de elemente bazate pe diferențele dintre evaluările acordate de alți utilizatori.

   În acest pas, ”deviațiile” se referă la diferența medie dintre evaluările date de utilizatori pentru perechile de elemente comune. Aceste deviații reflectă diferențele observate între evaluările utilizatorilor pentru aceste perechi.

   Spre exemplu, presupunem că există doi utilizatori, utilizatorul A și utilizatorul B, și două elemente, elementul X și elementul Y. Ambii utilizatori au evaluat ambele perechi de elemente. Deviația dintre elementul X și elementul Y reprezintă diferența medie între evaluările utilizatorului A și evaluările utilizatorului B pentru aceste elemente comune.

   Astfel, dacă utilizatorul A a evaluat elementul X cu nota 4 și elementul Y cu nota 3, iar utilizatorul B a evaluat elementul X cu nota 5 și elementul Y cu nota 2, deviația dintre elementul X și elementul Y este calculată ca diferența medie a acestor evaluări:

$$\frac{(4-5)+(3-2)}{2} = -0.5. \tag{2.2}$$

Această deviație indică faptul că, în medie, utilizatorul A acordă o evaluare cu 0.5 puncte mai mică pentru elementul Y în comparație cu elementul X.

3. **Prezicerea evaluărilor**: Odată ce deviațiile între perechile de elemente sunt calculate, modelul utilizează aceste deviații pentru a prezice evaluările utilizatorilor. Pentru un utilizator țintă, modelul ia în considerare evaluările sale anterioare și deviațiile asociate cu elementele evaluate de acesta.

4. **Agregarea și recomandarea**: Pentru a genera recomandări, algoritmul agregă evaluările prezise prin calcularea mediei evaluărilor prezise pentru fiecare element. Elementele sunt apoi clasate în funcție de evaluările prezise, iar elementele cu cele mai înalte evaluări sunt recomandate utilizatorului.

2.4.4 Modelul de grupare

Modelul de grupare[2], în eng. *Co-Clustering*, este un algoritm utilizat în sistemele de recomandare pentru a genera recomandări bazate pe grupuri de utilizatori și elemente. Acesta se bazează pe ideea că există anumite grupuri de utilizatori cu preferințe similare și anumite grupuri de elemente care sunt preferate în mod similar de acești utilizatori.

Modul de funcționare al acestui model este descris în cadrul următorilor pași:

1. **Reprezentarea interacțiunilor**: Primul pas constă în reprezentarea interacțiunilor utilizator-element într-o matrice. Rândurile matricei reprezintă utilizatorii, iar coloanele reprezintă elementele. Celulele matricei conțin evaluările sau feedback-ul implicit furnizat de utilizatori pentru elemente. În cazul în care un utilizator nu a evaluat un element, celula rămâne goală.

2. **Gruparea**: Algoritmul de grupare partiționează atât utilizatorii, cât și elementele, în grupuri, pe baza similarității lor în ceea ce privește tiparele de evaluare. Acest pas ajută la identificarea subgrupurilor omogene de utilizatori și elemente, ceea ce contribuie la îmbunătățirea calității recomandărilor.

3. **Inițializare**: Algoritmul de grupare inițializează grupurile în mod aleator sau folosind o strategie predefinită. Fiecare utilizator și element sunt atribuite inițial unui grup specific.

4. **Optimizare alternativă**: Algoritmul de grupare efectuează o optimizare alternativă pentru a rafina grupurile și a îmbunătăți acuratețea modelului. Aceasta implică două etape principale:

    - **Alocarea utilizatorilor în grupuri**: În această etapă, algoritmul atribuie utilizatorii în grupuri în funcție de similaritatea tiparelor lor de evaluare.

Scopul este de a *maximiza* similaritatea *în cadrul fiecărui grup*, în timp ce se *minimizează* similaritatea *între grupuri*.

- **Alocarea elementelor în grupuri**: Similar, algoritmul atribuie elementele în grupuri în funcție de tiparele lor de evaluare. Scopul este de a *maximiza* similaritatea *în cadrul fiecărui grup* de elemente, în timp ce se *minimizează* similaritatea *între grupuri*.

Aceste etape sunt repetate până când algoritmul atinge o soluție de grupare stabilă.

5. **Generarea recomandărilor**: Odată finalizat pasul de grupare, modelul generează recomandări pentru un utilizator dat. Acesta ia în considerare atât recomandările bazate pe utilizatori, cât și cele bazate pe elemente.

   - **Recomandări bazate pe utilizatori**: Modelul identifică grupul din care face parte utilizatorul țintă și recomandă elemente populare printre ceilalți utilizatori din grup. Aceste recomandări se bazează pe presupunerea că utilizatorii din același grup au preferințe similare.
   - **Recomandări bazate pe elemente**: Modelul identifică grupul din care face parte elementul țintă și recomandă elemente similare acestuia pe baza preferințelor utilizatorilor care au evaluat ambele elemente. Aceste recomandări se bazează pe presupunerea că utilizatorii care au evaluat elemente similare au preferințe similare.

2.4.5 Descompunerea valorilor singulare

Modelul de descompunere a valorilor singulare, în eng. *Singular Value Decomposition* (SVD), este un algoritm utilizat în sistemele de recomandare bazate pe filtrarea colaborativă. Acesta se bazează pe o descompunere matriceală a matricei de evaluări utilizator-element și utilizează valorile singulare pentru a extrage caracteristicile latente ale utilizatorilor și elementelor.

Modul de funcționare al acestui model este descris în cadrul următorilor pași:

1. **Reprezentarea datelor**: Datele de interacțiune utilizator-element sunt reprezentate sub forma unei matrice. Fie F matricea de evaluări, unde $F(i, j)$ reprezintă evaluarea utilizatorului i pentru elementul j.

2. **Descompunerea valorilor singulare**: Matricea de evaluări F este descompusă în produsul a trei matrici:

$$F = U \times \Sigma \times V^T \tag{2.3}$$

U reprezintă matricea utilizatorilor, Σ este matricea diagonală a valorilor singulare, iar V^T reprezintă matricea elementelor transpusă.

3. **Redimensionare și reducere dimensională (opțional)**: Pentru a reduce dimensionalitatea datelor, matricea Σ este redimensionată prin eliminarea valorilor singulare mici sau irelevante. Prin păstrarea doar a primelor k valori singulare, obținem o aproximare a matricei de evaluări:

$$\hat{F} = \hat{U} \times \hat{\Sigma} \times \hat{V}^T \tag{2.4}$$

unde $\hat{U}$, $\hat{\Sigma}$ și $\hat{V}^T$ sunt matricele reduse.

Prin acest pas sunt urmărite:

- **Eliminarea valorilor mici sau irelevante din** Σ : Valorile singulare în matricea Σ sunt ordonate descrescător și pot varia în magnitudine. Eliminarea valorilor singulare mici sau irelevante poate ajuta la reducerea zgomotului și a informațiilor irelevante în date, păstrând în același timp informațiile relevante.
- **Păstrarea a** k **celor mai relevante elemente**: Prin păstrarea doar a primelor k valori singulare și matricile corespunzătoare ($\hat{U}$, $\hat{\Sigma}$ și $\hat{V}^T$), putem obține o aproximare a matricei de evaluări inițiale cu o dimensiune mai mică. Aceasta reduce memoria necesară pentru stocarea datelor și poate accelera calculele ulterioare.

4. **Generarea recomandărilor**: Pe baza descompunerii reduse, recomandările sunt generate în funcție de produsul $U \times \Sigma$, sau $\hat{U} \times \hat{\Sigma}$ în cazul în care a fost executat și pasul 3. Pentru un utilizator dat, estimează rating-urile pentru elementele lipsă sau necunoscute utilizând produsul matriceal pentru rândul corespunzător utilizatorului.

Capitolul 3

Cum evaluăm un sistem de recomandare

3.1 Rădăcina erorii mediei pătratice

Rădăcina erorii mediei pătratice (*în eng. Root Mean Squared Error - RMSE*) este o metodă frecvent utilizată pentru a evalua precizia predicțiilor unui sistem de recomandare.[8]

Calculul RMSE implică următorii pași:

1. **Calculul diferențelor pătrate:** Pentru fiecare predicție, diferența dintre ratingul prezis și ratingul real este ridicată la pătrat.
2. **Calculul mediei:** Diferențele calculate anterior sunt apoi mediate pentru toate predicțiile.
3. **Calculul radicalului pătrat:** Se calculează radicalul pătrat al mediei diferențelor pătrate pentru a obține scorul RMSE.

Scorul RMSE ne arată cât de apropiate sunt ratingurile prezise de cele reale. Cu cât valoarea RMSE este mai mică, cu atât predicțiile sunt mai precise și, astfel, există o deviație mai mică între ratingurile prezise și cele reale.

Formula de calcul a metricii RMSE este următoarea;

$$\text{RMSE} = \sqrt{\frac{\sum_{p=1}^{N}(\text{rating prezis(p)} - \text{rating real(p)})^2}{N}} \tag{3.1}$$

3.2 Eroarea medie absolută

Eroarea medie absolută (*în eng. Mean Absolute Error - MAE*) este o altă măsură utilizată în sistemele de recomandare pentru a evalua precizia predicțiilor. Aceasta măsoară media valorilor absolute ale diferențelor dintre ratingurile prezise și ratingurile reale.[8]

Calculul MAE implică următorii pași:

1. **Calculul diferențelor absolute**: Se calculează diferența absolută dintre ratingul prezis și ratingul real pentru fiecare predicție, .
2. **Calculul mediei**: Diferențele absolute sunt apoi mediate pentru toate predicțiile.

Scorul MAE oferă o măsură a diferenței mediei absolute între ratingurile prezise și cele reale. Astfel, cu cât valoarea MAE este mai mică, cu atât predicțiile sunt considerate mai precise, deoarece există o discrepanță mai mică între ratingurile prezise și cele reale.

Formula de calcul a metricii MAE este următoarea:

$$\text{MAE} = \frac{\sum_{p=1}^{N} |\text{rating prezis(p)} - \text{rating real(p)}|}{N} \tag{3.2}$$

3.3 Evocare (recall)

În contextul sistemelor de recomandare, metrica denumită **evocare**, în eng. *recall*, măsoară proporția elementelor relevante ce au fost recomandate cu succes din numărul total al elementelor relevante existente pentru un utilizator.[8]

Metrica de evocare ia, în mod obișnuit, în considerare doar primele K elemente din setul de predicții, unde K reprezintă numărul de recomandări sau predicții făcute de către sistem.

Rațiunea din spatele considerării primelor K elemente este de a măsura cât de bine se descurcă sistemul în captarea celor mai relevante elemente. Astfel, prin concentrarea pe primele K recomandări, această metrică evidențiază abilitatea sistemului de a identifica și prezenta elementele relevante din cadrul unui set limitat de recomandări.

O valoare ridicată a acestei metrici indică faptul că sistemul de recomandare reușește să identifice o parte semnificativă din elementele relevante pentru utilizator.

În această lucrare, elementul relevant este considerat rating-ul pentru care diferența dintre valoarea reală și cea prezisă este mai mică de 0.5. Astfel, se iau în considerare doar predicțiile care au o discrepanță minimă față de preferințele reale ale utilizatorului.

Formula de calcul pentru metrica de evocare este următoarea [9]:

$$\text{Evocare(u, K)} = \frac{\text{Rating-uri relevante(u) in top K}}{\text{Rating-uri relevante(u)}} \tag{3.3}$$

$$\text{Scor Evocare(K)} = \frac{\sum_{\text{u} \in \text{TestSet}} Evocare(u, K)}{\text{Nr total predictii}} \tag{3.4}$$

3.4 Precizie

În contextul sistemelor de recomandare, **precizia** este o metrică ce măsoară acuratețea și relevanța elementelor recomandate. Ea cuantifică proporția elementelor recomandate care sunt cu adevărat relevante pentru preferințele sau nevoile utilizatorului. [8]

Precizia, în eng. *precision*, reprezintă pentru un sistem de recomandare o metrică care se concentrează asupra capacității sistemului de a oferi recomandări precise și relevante. Aceasta se asigură că elementele recomandate sunt în strânsă legătură cu preferințele și interesele utilizatorului.

Precizia în top K este o variantă specifică a metricii de precizie, utilizată în evaluarea sistemelor de recomandare. Aceasta se referă la măsurarea acurateții și relevanței primelor K elemente recomandate. În esență, se calculează raportul dintre numărul de elemente relevante recomandate corect din primele K poziții și numărul total de elemente recomandate în acele poziții.

Elementele relevante sunt definite în prezentarea metricii de evocare, regula fiind păstrată și pentru metrica de precizie.

Un scor de precizie ridicat indică faptul că sistemul reușește să recomande o parte semnificativă de elemente relevante pentru utilizator.

Optimizând pentru precizie, sistemele de recomandare își propun să minimizeze recomandările irelevante sau nedorite, îmbunătățind experiența utilizatorului și crescând probabilitatea de satisfacție a acestuia.

Astfel, această metrică ajută la reducerea suprasolicitării informaționale și la personalizarea recomandărilor în funcție de preferințele și nevoile specifice ale fiecărui utilizator.

Formula de calcul pentru metrica de precizie este următoarea [9]:

$$\text{Scor de precizie(K)} = \frac{\sum_{\text{u} \in \text{TestSet}} \text{Nr iteme relevante in top K(u)}}{\text{Nr total iteme in top K}} \tag{3.5}$$

3.5 Scorul F1

În contextul sistemelor de recomandare, scorul F1 este utilizat pentru a măsura calitatea recomandărilor furnizate de sistem. În mod specific, acesta analizează atât numărul

de recomandări relevante corecte (precizie), cât și capacitatea sistemului de a recupera toate elementele relevante disponibile (evocare).[8].

Calcularea scorului F1 la K implică determinarea preciziei și a evocării la K și combinarea acestora utilizând formula specifică. Astfel, metrica F1 la K oferă o evaluare mai echilibrată a performanței sistemului de recomandare, având în vedere atât corectitudinea recomandărilor, cât și capacitatea de a recupera toate elementele relevante.

Un scor F1 ridicat indică faptul că sistemul de recomandare furnizează recomandări precise și acoperă o proporție semnificativă a elementelor relevante disponibile. Acest lucru sugerează că utilizatorii vor primi recomandări relevante și complete, fapt ce poate îmbunătăți experiența lor de utilizare.

Formula de calcul pentru scorul F1 este următoarea [9]:

$$\text{F1 Score(K)} = \frac{2 * \text{Precizie(K)} * \text{Evocare(K)}}{\text{Precizie(K)} + \text{Evocare(K)}} \tag{3.6}$$

3.6 Robustețe

În contextul sistemelor de recomandare, metrica de robustețe măsoară capacitatea unui algoritm de a păstra sau îmbunătăți performanța sa în situații neașteptate.

O metodă de evaluare a robusteții este de a simula un scenariu în care un număr specific de utilizatori acordă aceleași evaluări identice pentru aceleași articole și, ulterior, adăugați acești utilizatori în setul de date de antrenament.

Atunci când setul de date este împărțit în setul de antrenament și setul de testare, adăugarea acestor utilizatori cu evaluări identice în setul de antrenament poate afecta capacitatea algoritmului de a genera predicțiile în mod corespunzător. În această situație, algoritmul trebuie să fie capabil să recunoască și să ignore influența excesivă a acestor utilizatori lipsiți de preferințe.

Măsurarea robusteții implică următorii pași:

1. **Împărțirea setului de date în set de antrenament și set de testare**
2. **Crearea unui număr dat de utilizatori**, numărul de utilizatori fiind luat din interfața aplicației
3. **Pentru fiecare utilizator, crearea intrărilor pentru fiecare element din setul de date**, rating-ul intrărilor fiind luat din interfața aplicației
4. **Inserarea utilizatorilor fictivi în setul de antrenament**
5. **Antrenarea modelului pe setul de antrenament modificat**

6. **Generarea predicțiilor și evaluarea lor**, evaluare realizată folosind, la alegere, un set de metrici de bază:
    - RMSE și MAE
    - Precizie, Evocare, F1

Prin evaluarea robusteții, se poate observa dacă un algoritm este capabil să mențină performanța sa în fața schimbărilor neașteptate în datele de antrenament, precum și să ofere recomandări relevante chiar și atunci când se confruntă cu situații atipice.

3.7 Deficitul de date

În contextul sistemelor de recomandare, metrica deficitului de date, în eng. *data scarcity*, se referă la evaluarea capacității unui algoritm de a face recomandări precise și relevante în situații în care există o cantitate limitată de date disponibile pentru antrenare.

Măsurarea deficitului de date implică următorii pași:

1. **Împărțirea setului de date într-un set de antrenament și un set de testare**
2. **Setul de date de antrenament este împărțit în mai multe sub-seturi**
3. **În fiecare etapă, câte un sub-set este adăugat la setul de antrenament**, iar modelul de recomandare este reantrenat folosind noile date adăugate
4. **După fiecare reantrenare, modelul este evaluat pe setul de testare** pentru a determina cât de bine se descurcă în generarea predicțiilor. Acest proces este repetat pentru fiecare sub-set adăugat, iar evaluarea este realizată folosind, la alegere, un set de metrici de bază:
    - RMSE și MAE
    - Precizie, Evocare, F1

Scopul acestui exercițiu este de a observa cum evoluează performanța modelului de recomandare pe măsură ce sunt adăugate mai multe sub-seturi în antrenarea sa. Metrica de deficit de date explorează cum modelul se adaptează și se îmbunătățește sau își păstrează performanța în situații de date reduse.

Evaluarea acestui aspect este importantă deoarece reflectă capacitatea modelului de a face față situațiilor în care datele sunt limitate sau insuficiente, ceea ce poate fi o problemă în scenariile reale. Prin măsurarea calității recomandărilor pe măsură ce sunt adăugate sub-seturi în mod treptat, se poate obține o imagine mai clară a capacității de adaptare și de performanță a modelului în condițiile deficitului de date.

Capitolul 4

Seturi de date și biblioteci utilizate

4.1 Seturile de date

Seturile de date **MovieLens 100K** și **MovieLens 1M**[5] sunt utilizate pentru a testa și evalua performanța algoritmilor de recomandare.

Principalele diferențe dintre cele două seturi ar fi următoarele:

- **Mărimea datelor**:
  - **MovieLens 100K**: Conține aproximativ 100.000 de evaluări de filme realizate de aproximativ 1000 de utilizatori pentru o colecție de circa 1700 de filme. Este un set de date mai mic, care poate fi mai ușor de gestionat și de procesat
  - **MovieLens 1M**: Conține aproximativ 1.000.000 de evaluări, de la mai mult de 6000 de utilizatori pentru peste 4000 de filme. Ca urmare, acest set de date este mai mare și mai complex din punct de vedere al volumului de date
- **Detalii despre filme**:
  - **MovieLens 100K**: Informațiile despre filme din acest set de date sunt relativ simple și se concentrează pe titlu, genuri și data de lansare
  - **MovieLens 1M**: Sunt oferite informații mai detaliate, precum informații despre regizori, actori, și alte caracteristici suplimentare
- **Utilizatori și evaluări**:
  - **MovieLens 100K**: Oferă o imagine mai mică a comportamentului utilizatorilor, deoarece implică un număr mai mic de evaluări și utilizatori.
  - **MovieLens 1M**: Datorită dimensiunii mai mari, setul de date MovieLens 1M furnizează o perspectivă mai complexă asupra preferințelor utilizatorilor și modelelor de comportament.

- **Aplicații ale seturilor de date**:
  - **MovieLens 100K**: Poate fi preferat în situațiile în care se dorește o evaluare rapidă a algoritmilor pe un set mai mic de date
  - **MovieLens 1M**: Este util atunci când se doresc teste mai cuprinzătoare și analize mai detaliate pe seturi de date mai mari

4.2 Bibliotecile

Scopul bibliotecile este de a oferi dezvoltatorilor un set de funcționalități, instrumente și componente predefinite pentru a rezolva anumite probleme sau sarcini specifice în dezvoltarea software.

Prin urmare, utilizarea lor facilitează:

- **Economisirea de timp și efort**, evitând necesitatea de a rescrie cod de la zero pentru fiecare proiect
- **Îmbunătățirea calității codului**, încurajând urmărirea bunelor practici în dezvoltarea software

Voi descrie astfel, în continuarea lucrării, bibliotecile utilizate în cadrul acestui proiect, alături de motivul alegerii lor.

4.2.1 Surprise

Surprise[6] este o bibliotecă Python specializată în dezvoltarea și evaluarea modelelor de recomandare. A fost creată pentru a simplifica procesul de construire a algoritmilor de recomandare și pentru evaluarea acestora într-un mod convenabil și eficient.

Am ales sa folosesc această bibliotecă deoarece oferă următoarele avantaje:

1. **Seturi de date integrate în bibliotecă**: seturile de date MovieLens sunt integrate în această bibliotecă, devenind mai ușor de accesat
2. **Modelele de recomandare implementate**: în această bibliotecă se regăsesc o multitudine de modele de recomandare deja implementate, dintre care am ales sa integrez în aplicația mea următoarele modele, detaliate anterior în secțiunea 2.4:
   - Baseline
   - K-Nearest Neighbours
   - Slope One

- Co-Clustering
- SVD

4.2.2 Flask

Flask[3] este un cadru (*în eng. framework*) de dezvoltare pentru aplicații web în Python. El furnizează un set de instrumente și structuri ce fac dezvoltarea de aplicații web mai ușoară și mai eficientă.

Flask este cunoscut pentru simplitatea sa și pentru abordarea sa minimalistă, permițând dezvoltatorilor să construiască aplicații web rapide și flexibile.

Am ales să folosesc această bibliotecă deoarece oferă următoarele avantaje:

1. **Permite dezvoltarea cu ușurință a aplicațiilor web în limbajul Python**
2. **Inserarea și editarea informațiilor din paginile de HTML sunt realizate cu ușurință**, folosind motorul de șabloane **Jinja**, integrat în această bibliotecă
3. **Oferă posibilitatea de a defini rute personalizate pentru diverse cereri HTTP (GET, POST, etc.).** Astfel, am separat aplicația în două servere, anume într-un server de front-end ce face apeluri către un server de API-uri pentru antrenarea și evaluarea modelelor.

4.2.3 Gunicorn

Însă, deoarece biblioteca Flask a fost proiectată să funcționeze în mod sincron, adică se așteaptă rezultatul unui apel către un API pentru a putea începe executarea unei alte funcții, am ales să folosesc biblioteca **Gunicorn**[4] ce utilizează un model de lucru asincron.

Astfel, pot fi gestionate mai multe cereri HTTP în paralel, fără a bloca procesul principal. Acest lucru permite serverului să fie mai receptiv și să poată răspunde rapid la cereri multiple.

4.2.4 Plotly

Plotly[7] este o bibliotecă interactivă și open-source pentru crearea de grafice și vizualizări de date în limbajul de programare Python, dar și în alte limbaje de programare precum R, JavaScript și Julia.

Astfel, am folosit această bibliotecă pentru afișarea rezultatelor obținute în urma evaluării modelelor de recomandare, deoarece permite generarea graficelor de diferite tipuri, alături de moduri interactive de vizualizare a datelor, precum selectarea unei zone de interes în grafic sau izolarea elementelor de interes prin eliminarea a celorlalte elemente.

4.2.5 Pandas

Pandas[14] este o bibliotecă open-source în limbajul de programare Python, specializată în analiza și manipularea datelor. A fost dezvoltată pentru a oferi un set puternic de instrumente și funcționalități pentru lucrul cu date structurate, permițând dezvoltatorilor să efectueze operații complexe de manipulare, transformare și analiză a datelor într-un mod eficient și convenabil.

Astfel, am folosit această bibliotecă deoarece oferă un obiect asemănător unor tabele de date, numit *DataFrame*. Acest aspect este important deoarece seturile de date se regăsesc, de obicei, în format CSV, iar acest obiect permite citirea și prelucrarea lor în limbajul Python. Prelucrarea seturilor de date este importantă în metricile de robustețe și de deficit de date, unde seturile de antrenament sunt modificate în scopul metricilor.

4.2.6 Bootstrap

Bootstrap[1] este un cadru (*în eng. framework*) de front-end sau o bibliotecă CSS și JavaScript open-source, dezvoltată de către Twitter, ce furnizează șabloane, componente și stiluri predefinite.

Cu Bootstrap, dezvoltatorii pot să se concentreze mai mult pe funcționalitatea aplicației și mai puțin pe designul detaliat al interfeței.

Astfel, am folosit acest framework pentru a îmbunătăți aspectul aplicației, utilizând componente predefinite din cadrul său, pentru a crea o interfață atractivă și adaptabilă la diferite rezoluții.

Capitolul 5

Implementarea aplicației

Cel mai simplu mod în care un utilizator poate interacționa și vizualiza rezultatele acțiunilor sale este o interfață web. Astfel, pentru scopul acestei lucrări am ales implementarea unei aplicații web.

Aplicația este formată din două servere, unul ce afișează atât interfața de bază a aplicației cât și graficele, și celălalt ce se ocupă de antrenarea și evaluarea modelelor, ce are ca scop returnarea rezultatelor în urma aplicării metricilor de evaluare. Comunicarea între cele două servere este realizată prin intermediul cererilor de tip HTTP folosind verbe precum GET sau POST, conform arhitecturii RESTful[15].

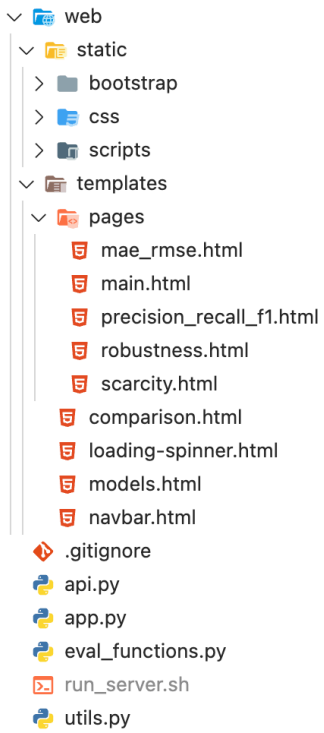

Figura 5.1: Structura proiectului

Structura proiectului ce conține fișierele aplicației web este următoarea:

- În directorul `static` sunt aflate fișierele ce sunt refolosite în cadrul proiectului, precum:
  1. fișierele `bootstrap`, atât CSS cât și JavaScript
  2. fișierele de stilizare din directorul `css`
  3. script-urile comune din directorul `scripts`
- În directorul `templates` avem componentele HTML ce sunt utilizate de mai multe ori în cadrul aplicației
- În subdirectorul `pages` avem paginile pentru fiecare dintre metricile de evaluare. Aceste pagini HTML conțin, în JavaScript, și script-urile aferente fiecăreia dintre ele.
- În fișierul `utils.py` sunt scrise funcțiile utilitare din cadrul proiectului, precum cele de validare în backend a valorilor introduse, cât și cele de formare a aspectului valorilor din grafice
- În fișierul `eval_functions.py` sunt scrise metricile de evaluare pentru evocare, precizie si pentru scorul F1. Acest fișier este folosit în cadrul proiectului precum o bibliotecă.
- În fișierul `api.py` avem codul aferent server-ului de API-uri
- În fișierul `app.py` avem codul aferent server-ului de frontend, ce oferă paginile de HTML și graficele din acestea

5.1 Server backend pentru antrenarea și evaluarea modelelor

În acest server sunt localizate API-urile ce antrenează și evaluează modelele de recomandare. Accesarea lor se face utilizând apeluri HTTP de tip POST, prin care sunt transmiși parametrii necesari fiecărei metrici de evaluare, iar ca răspuns este returnat un document JSON(JavaScript Object Notation) ce conține datele ce trebuie afișate în graficele din interfață.

Lista API-urilor create:

1. Eroarea medie absoluta și rădăcina erorii mediei pătratice
2. Precizie, evocare și scorul F1

3. Robustețe

4. Deficit de date

Toate aceste API-uri au următorii parametrii comuni, necesari pentru antrenarea modelelor:

1. **Numele modelului ales** din lista de modele enumerate în secțiunea 2.4

2. **Valoarea seed-ului pentru impărțirea setului de date**, deoarece trebuie să împărțim în același mod setul de date pentru fiecare model, astfel încât acestea să fie antrenate pe aceleași date

3. **Setul de date ales**, anume MovieLens 100K sau MovieLens 1M

5.1.1 Eroarea medie absolută și rădăcina erorii mediei pătratice

În acest API sunt implementate metricile descrise în subsecțiunile 3.1 și 3.2. De altfel, o altă valoare măsurată în acest API este durata timpului de antrenare și testare a modelelor alese.

Implementarea acestui API a fost realizată în felul următor:

```
@app.route("/api/v1/mae_rmse", methods=["POST"])
def post_mae_rmse_model():
    try:
        request_data = request.json

        model_name = request_data.get("model_name")
        random_state_value = int(request_data.get("random_state_value
    "))
        selected_dataset = request_data.get("dataset")

    except Exception as exception:
        return jsonify({"Error": str(exception)}), 400

    model = utils.get_model_instance(model_name)
    selected_dataset = utils.get_dataset(selected_dataset)

    data = Dataset.load_builtin(selected_dataset)
    train_data, test_data = train_test_split(
        data, train_size=0.8, random_state=random_state_value
    )
```

```
20
21      start_time = time.time()
22      model.fit(train_data)
23
24      predictions = model.test(test_data)
25      end_time = time.time()
26
27      mae_score = mae(predictions, verbose=False)
28      rmse_score = rmse(predictions, verbose=False)
29
30      response_data = {
31          "rmse_score": rmse_score,
32          "mae_score": mae_score,
33          "execution_time": end_time - start_time,
34      }
35
36      return jsonify(response_data)
```

5.1.2 Precizie, evocare și scorul F1

În acest API sunt implementate metricile descrise în subsecțiunile 3.3, 3.4 și 3.5. Acest API necesită un parametru în plus față de parametrii comuni, anume **K**, ce reprezintă numărul de rating-uri evaluate pentru fiecare utilizator, în urma sortării rating-urilor acestuia în ordine descrescatoare. Acest concept mai este cunoscut si drept Top K.

Funcțiile de evaluare au fost implementate după cum urmează:

1. **Evocare**:

```
1  def recall_score(test_set, predictions, k=10):
2      set_of_users = {user[0] for user in test_set}
3      total_recall = 0
4
5      for user in set_of_users:
6          list_of_predictions = [
7              prediction for prediction in predictions if
       prediction.uid == user
8          ]
9          sorted_predictions = sorted(
10             list_of_predictions, key=lambda x: x.est, reverse=
       True
11         )
```

```
        top_k_predictions = sorted_predictions[:k]

        relevant_items_count = sum(
            1
            for prediction in top_k_predictions
            if abs(prediction.r_ui - prediction.est) >= 0.5
        )
        total_relevant_items = sum(
            1
            for prediction in list_of_predictions
            if abs(prediction.r_ui - prediction.est) >= 0.5
        )

        if total_relevant_items > 0:
            recall = relevant_items_count / total_relevant_items
            total_recall += recall

    return total_recall / len(set_of_users)

```

2. **Precizie**:

```
def precision_score(test_set, predictions, k=10):
    set_of_users = {user[0] for user in test_set}
    total_precision = 0

    for user in set_of_users:
        list_of_predictions = [
            prediction for prediction in predictions if
   prediction.uid == user
        ]
        sorted_predictions = sorted(
            list_of_predictions, key=lambda x: x.est, reverse=
   True
        )
        top_k_predictions = sorted_predictions[:k]

        relevant_items_count = sum(
            1
            for prediction in top_k_predictions
            if abs(prediction.r_ui - prediction.est) >= 0.5
        )
```

```
        precision = relevant_items_count / k
        total_precision += precision

    return total_precision / len(set_of_users)

```

3. **Scor F1**, însă întrucât această metrică face parte din setul de metrici 'precizie, evocare, f1' și formula ei se bazează pe valorile a celorlalte două metrici, am ales să folosesc o singură funcție ce returnează toate cele trei valori:

```
def precision_recall_f1(test_set, predictions, k=10):
    precision = precision_score(test_set, predictions, k)
    recall = recall_score(test_set, predictions, k)

    if precision + recall == 0:
        f1 = 0
    else:
        f1 = 2 * (precision * recall) / (precision + recall)

    return precision, recall, f1

```

Implementarea acestui API a fost realizată în felul următor:

```
@app.route("/api/v1/precision_recall_f1", methods=["POST"])
def post_precision_recall_f1_model_k():
    try:
        request_data = request.json

        model_name = request_data.get("model_name")
        random_state_value = int(request_data.get("random_state_value
    "))
        k = int(request_data.get("k"))
        selected_dataset = request_data.get("dataset")

    except Exception as exception:
        return jsonify({"Error": str(exception)}), 400

    selected_dataset = utils.get_dataset(selected_dataset)

    data = Dataset.load_builtin(selected_dataset)

    train_data, test_data = train_test_split(
```

```
        data, train_size=0.8, random_state=random_state_value
    )

    model = utils.get_model_instance(model_name)
    model.fit(train_data)

    predictions = model.test(test_data)

    precision_score, recall_score, f1_score = eval_func.
   precision_recall_f1(
        test_data, predictions, k
    )

    response_data = {
        "precision_score": precision_score,
        "recall_score": recall_score,
        "f1_score": f1_score,
    }

    return jsonify(response_data)
```

5.1.3 Robustețe

În acest API este implementată metrica de robustețe descrisă în subsecțiunea 3.6, astfel că are nevoie de următorii parametrii, conform modului său de funcționare:

1. **Numărul de utilizatori** ce trebuie adăugați la setul de antrenament
2. **Rating-ul** pentru toate intrările utilizatorilor adăugați
3. **Metoda de comparare**, deoarece evaluarea rezultatelor în urma adăugării utilizatorilor este realizată folosind unul dintre următoarele seturi de metrici:
    - MAE și RMSE
    - Precizie, evocare, scor F1. Pentru această opțiune este necesar și parametrul K.

Implementarea acestui API a fost realizată în felul următor:

```
@app.route("/api/v1/robustness", methods=["POST"])
def post_robustness():
    try:
        request_data = request.json
```

```
5
6           model_name = request_data.get("model_name")
7           nr_users = int(request_data.get("nr_users"))
8           rating = float(request_data.get("rating"))
9           random_state_value = int(request_data.get("random_state_value
    "))
10          comparison_method = request_data.get("comparison_method")
11          k_value = int(request_data.get("k"))
12          selected_dataset = request_data.get("dataset")
13
14          if comparison_method is None:
15              raise KeyError(
16                  "comparison_method needs to be specified in the
    request body"
17              )
18
19          if comparison_method == "prf" and k_value is None:
20              raise KeyError(
21                  "k needs to be specified for the precision, recall
    and f1 metrics"
22              )
23
24      except Exception as e:
25          return jsonify({"Error": str(e)}), 400
26
27      selected_dataset = utils.get_dataset(selected_dataset)
28
29      data = Dataset.load_builtin(selected_dataset)
30
31      train_data, test_data = train_test_split(
32          data, train_size=0.8, random_state=random_state_value
33      )
34
35      train_data = pd.DataFrame(train_data.build_testset())
36      data = pd.DataFrame(data.build_full_trainset().all_ratings())
37      reader = Reader(rating_scale=(data[2].min(), data[2].max()))
38
39      if rating > data[2].max() or rating < data[2].min():
40          return jsonify(
41              {
```

```
                "Error": f"rating needs to be in the dataset's ratings interval: [{data[2].min()}:{data[2].max()}]"
            }
        )

    item_column = data.columns.tolist()[1]
    items_list = data.sort_values(by=item_column)[item_column].unique()

    def create_user(nr_votes, rating):
        user_id = int(data[0].max() + 1)
        user_rated_items = items_list[:nr_votes]

        user_ids = [user_id] * nr_votes

        user_ratings = pd.DataFrame(columns=data.columns)
        user_ratings[0] = user_ids
        user_ratings[1] = user_rated_items
        user_ratings[2] = rating

        return user_ratings

    for _ in range(nr_users):
        train_data = pd.concat(
            [train_data, create_user(len(items_list), rating)], ignore_index=True
        )

    train_data = Dataset.load_from_df(train_data, reader=reader)
    train_data = train_data.build_full_trainset()

    model = utils.get_model_instance(model_name)

    model.fit(train_data)

    predictions = model.test(test_data)

    response_body = {"rating": rating, "nr_users": nr_users}

    if comparison_method == "mae_rmse":
        mae_score = mae(predictions, verbose=False)
```

```
        rmse_score = rmse(predictions, verbose=False)

        response_body["mae_score"] = mae_score
        response_body["rmse_score"] = rmse_score

    elif comparison_method == "prf":
        precision_score, recall_score, f1_score = eval_func.
    precision_recall_f1(
            test_data, predictions, k_value
        )

        response_body["precision_score"] = precision_score
        response_body["recall_score"] = recall_score
        response_body["f1_score"] = f1_score

    return jsonify(response_body)
```

5.1.4 Deficit de date

În acest API este implementată metrica pentru deficitul de date, descrisă în subsecțiunea 3.7. În afara parametrilor comuni, această metrică mai are nevoie de următorii parametrii:

1. **Numărul de sub-seturi** în care să fie împărțit setul de antrenament
2. **Metoda de comparare**, deoarece evaluarea rezultatelor în urma reantrenării după adăugarea sub-seturilor este realizată folosind unul dintre următoarele seturi de metrici:
    - MAE și RMSE
    - Precizie, evocare, scor F1. Pentru această opțiune este necesar și parametrul K.

Implementarea acestui API a fost realizată în felul următor:

```
@app.route("/api/v1/scarcity", methods=["POST"])
def post_scarcity():
    try:
        request_data = request.json

        model_name = request_data.get("model_name")
        random_state_value = int(request_data.get("random_state_value
    "))
```

```
        comparison_method = request_data.get("comparison_method")
        k_value = int(request_data.get("k"))
        model_feeding_rate = int(request_data.get("model_feeding_rate
"))
        selected_dataset = request_data.get("dataset")

        if comparison_method is None:
            raise KeyError(
                "comparison_method needs to be specified in the
request body"
            )

        if comparison_method == "prf" and k_value is None:
            raise KeyError(
                "k needs to be specified for the precision, recall
and f1 metrics"
            )
    except Exception as e:
        return jsonify({"Error": str(e)}), 400

    selected_dataset = utils.get_dataset(selected_dataset)

    table = []
    data = Dataset.load_builtin(selected_dataset)

    trainset, testset = train_test_split(
        data, train_size=0.8, random_state=random_state_value
    )

    model = utils.get_model_instance(model_name)

    trainset_df = pd.DataFrame(trainset.build_testset())

    reader = Reader(rating_scale=(trainset_df[2].min(), trainset_df
[2].max()))

    len_trainset_df = len(trainset_df)

    for i in range(
        len_trainset_df // model_feeding_rate,
        len_trainset_df // model_feeding_rate + len_trainset_df,
```

```
        len_trainset_df // model_feeding_rate,
    ):
        partial_trainset = Dataset.load_from_df(trainset_df.head(i),
reader)
        partial_trainset = partial_trainset.build_full_trainset()
        model.fit(partial_trainset)
        predictions = model.test(testset)

        new_line = {
            "nr_items": i,
        }

        if comparison_method == "prf":
            precision_score, recall_score, f1_score = eval_func.
precision_recall_f1(
                testset, predictions, k_value
            )

            new_line["precision_score"] = precision_score
            new_line["recall_score"] = recall_score
            new_line["f1_score"] = f1_score

        elif comparison_method == "mae_rmse":
            new_line["mae_score"] = mae(predictions, verbose=False)
            new_line["rmse_score"] = rmse(predictions, verbose=False)

        table.append(new_line)

    return jsonify(table)
```

5.2 Server frontend

Acest server îndeplinește două obiective:

1. **Oferirea documentelor HTML drept pagini web**, accesibile prin anumite rute. În aceste pagini, componentele HTML reutilizabile sunt inserate folosind formatul oferit de motorul de șabloane Jinja (ex. `{% include 'navbar.html'%}`)

   Codul pentru transmiterea paginilor web:

```
@app.route("/")
```

```
def show_main():
    return render_template("/pages/mae_rmse.html")

@app.route("/mae_rmse")
def show_mae_rmse_graph():
    return render_template("/pages/mae_rmse.html")

@app.route("/precision_recall_f1")
def show_precision_f1_recall():
    return render_template("/pages/precision_recall_f1.html")

@app.route("/robustness")
def show_robustness_graph():
    return render_template("/pages/robustness.html")

@app.route("/scarcity")
def show_scarcity_graph():
    return render_template("/pages/scarcity.html")
```

2. **Încărcarea datelor în grafice**, acțiune pentru care este necesară comunicarea cu serverul de backend pentru a primi datele în urma evaluării modelelor. Pentru această acțiune sunt create endpoint-uri către care se fac apeluri de tip GET cu parametrii preluați din interfață paginii web, cu ajutorul limbajului JavaScript. Mai departe, din aceste endpoint-uri se fac cerere de tip POST, cu valorile primite de la frontend, către serverul de backend.

Codul Python pentru endpoint-ul ce se ocupa de popularea graficelor metricilor de evaluare MAE și RMSE:

```
@app.route("/mae_rmse/get_plots/<dataset>/<models>/", methods=["
    GET"])
def get_mae_rmse_plots(dataset, models):
    selected_models = models.split(",")

    mae_to_plot, rmse_to_plot, time_to_plot = utils.get_plots(
        selected_models, utils.get_mae_rmse_score_bars, dataset=
    dataset
    )

    fig_mae = go.Figure(data=mae_to_plot)
    fig_mae.update_layout(
        title="Eroarea medie absoluta (MAE)",
```

```
        barmode="group",
        yaxis_range=[
            math.log10(min(mae_to_plot, key=lambda bar: bar["y"])["y"][0] * 0.98),
            math.log10(max(mae_to_plot, key=lambda bar: bar["y"])["y"][0] * 1.02),
        ],
    )
    fig_mae.update_yaxes(type="log")

    fig_rmse = go.Figure(data=rmse_to_plot)
    fig_rmse.update_layout(
        title="Radacina erorii mediei patratice (RMSE)",
        barmode="group",
        yaxis_range=[
            math.log10(min(rmse_to_plot, key=lambda bar: bar["y"])["y"][0] * 0.98),
            math.log10(max(rmse_to_plot, key=lambda bar: bar["y"])["y"][0] * 1.02),
        ],
    )
    fig_rmse.update_yaxes(type="log")

    fig_time = go.Figure(data=time_to_plot)
    fig_time.update_layout(
        title="Timpul de antrenare si de predictie a modelelor",
        barmode="group",
        yaxis_range=[
            math.log10(min(time_to_plot, key=lambda bar: bar["y"])["y"][0] * 0.98),
            math.log10(max(time_to_plot, key=lambda bar: bar["y"])["y"][0] * 1.6),
        ],
    )
    fig_time.update_yaxes(type="log")

    return jsonify(
        {
            "mae_plot": fig_mae.to_plotly_json(),
            "rmse_plot": fig_rmse.to_plotly_json(),
            "time_plot": fig_time.to_plotly_json(),
```

```
        }
    )
```

Codul JavaScript ce se ocupă de apelarea acestui endpoint:

```
const fetchModels = async () => {
  try {
    const selectedDataset = document.getElementById("
    select_dataset").value;
    const modelsInPage = [...document.getElementsByName("model")
    ];
    const inputsInPage = [...document.getElementsByTagName("
    input")]
    const disableAllInputs = (() => {
      inputsInPage.map((input) => input.disabled = true)
    })();
    const filteredModelsInPage = (() => {
      if (modelsInPage.every((input) => !input.checked)) {
        return modelsInPage.map((input) => input.value)
      }
      return modelsInPage
        .filter((input) => !!input.checked)
        .map((input) => input.value);
    })();
    const response = await (async () => {
      if (!filteredModelsInPage.length) {
        return await fetch(`/mae_rmse/get_plots/${
    selectedDataset}`);
      }
      return await fetch(
        `/mae_rmse/get_plots/${selectedDataset}/${
    filteredModelsInPage.join(",")}/`
      );
    })();
    const data = await response.json();
    const enableAllInputs = (() => {
      inputsInPage.map((input) => input.disabled = false)
    })();
    return data;
  } catch (error) {
    console.error("Error fetching plot data:", error);
    const enableAllInputs = (() => {
```

```
      inputsInPage.map((input) => input.disabled = false)
    })();
    return null;
  }
};

document.getElementById("selectAll").addEventListener("change",
    (event) => {
  const checkboxes = document.querySelectorAll(
    'input[type="checkbox"][name="model"]'
  );
  for (let i = 0; i < checkboxes.length; i++) {
    checkboxes[i].checked = event.target.checked;
  }
});

const plotModels = async () => {
  document.getElementById("loading-window").style.display = "
    block";

  const data = await fetchModels();

  document.getElementById("loading-window").style.display = "
    none";

  if (!data) return;

  Plotly.newPlot("mae-plot", data.mae_plot);
  Plotly.newPlot("rmse-plot", data.rmse_plot);
  Plotly.newPlot("time-plot", data.time_plot);
};

window.addEventListener("load", async () => {
  setActiveMenu('mae_rmse')
  await plotModels();
});

document
  .getElementById("evaluate_button")
  .addEventListener("click", async (event) => {
    event.preventDefault();
```

```
    await plotModels();
  });
```

Codul pentru generarea elementelor vizuale din grafice:

```
def get_mae_rmse_score_bars(model_name, random_state_value, dataset):
    bar_width = 0.1

    post_data = {
        "model_name": model_name,
        "random_state_value": random_state_value,
        "dataset": dataset,
    }

    response = requests.post("http://127.0.0.1:8000/api/v1/mae_rmse", json=post_data)
    response_data = response.json()

    mae_score = response_data["mae_score"]
    rmse_score = response_data["rmse_score"]
    elapsed_time = response_data["execution_time"]

    mae_bar = go.Bar(
        name=f"{model_name}",
        x=["MAE"],
        y=[mae_score],
        width=bar_width,
        text=[f"{model_name} <br> MAE: {round(mae_score, 6)}"],
        textposition="outside",
        hoverinfo="text",
        hovertext=f"Model: {model_name} <br>MAE: {mae_score}",
    )
    rmse_bar = go.Bar(
        name=f"{model_name}",
        x=["RMSE"],
        y=[rmse_score],
        width=bar_width,
        text=[f"{model_name} <br> RMSE: {round(rmse_score, 6)}"],
        textposition="outside",
        hoverinfo="text",
```

```
        hovertext=f"Model: {model_name} <br>RMSE: {rmse_score}",
    )
    time_bar = go.Bar(
        name=f"{model_name}",
        x=["Time"],
        y=[elapsed_time],
        width=bar_width,
        text=[f"{model_name} <br> Time: {round(elapsed_time, 6)}
s"],
        textposition="outside",
        hoverinfo="text",
        hovertext=f"Model: {model_name} <br>Time: {round(
elapsed_time, 6)}s",
    )

    return mae_bar, rmse_bar, time_bar
```

Capitolul 6

Interfața aplicației

6.1 Structura

Interfața aplicației este formată din 4 pagini HTML în care metricile de evaluare fie sunt grupate fie sunt unice în pagină, și anume:

1. **MAE + RMSE**: această pagină prezintă rezultatele metricilor pentru eroarea medie absolută și pentru rădăcina erorii mediei pătratice.
2. **Precision, Recall, F1**: în această pagină sunt prezentate rezultatele metricilor de precizie, evocare și scor F1
3. **Robustness**: această pagină conține rezultatele metricii de robustețe
4. **Scarcity**: în această pagina sunt graficele pentru metrica deficitului de date

Paginile sunt asemănătoare între ele, astfel că următoarele componente sunt afișate pe fiecare dintre ele:

1. **Bara de navigare**: în bara de navigare sunt trecute rutele către cele 4 pagini cu metrici de evaluare

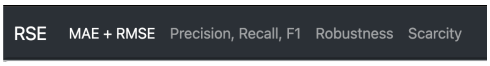

Figura 6.1: Bara de navigare

2. **Configurarea metricilor**: sub bara de navigare se află o componentă de tip acordeon ce conține câmpuri ce trebuie completate pentru metricile alese. Această componentă poate conține mai multe sau mai puține câmpuri, în funcție de pagina accesată. Parametrii principali sunt formați din lista de modele ce se doresc a fi comparate și setul de date dorit.

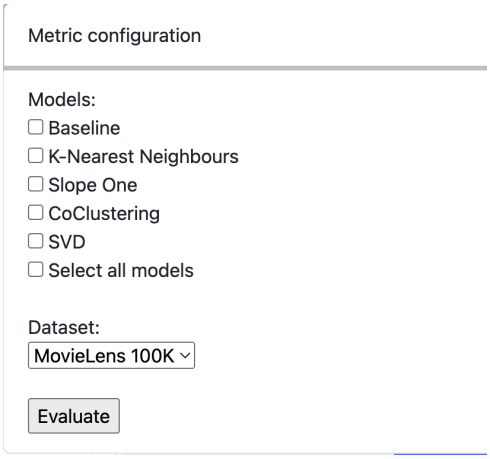

Figura 6.2: Configurarea metricilor

În afara acestora, în funcție de pagină, se regăsesc și următorii parametrii:

(a) **Precision, Recall, F1**: în această pagina mai regăsim un câmp ce trebuie completat cu valoarea dorită pentru parametrul K specific acestor metrici

(b) **Robustness**: având în vedere modul de funcționare al metricii, în această pagină se află câmpuri ce trebuie completate cu valori pentru numărul de utilizatori ce se dorește a fi adăugați, rating-ul ce trebuie să îl aibă intrarile acestor utilizatori și metoda de comparare aleasă(în cazul în care este ales setul 'precision, recall, f1', mai apare un câmp de completat pentru parametrul K)

(c) **Scarcity**: pentru această metrică sunt câmpuri pentru numărul de subset-uri în care să fie divizat setul de antrenament și pentru metoda de comparare aleasă (unde, precum în pagina Robustness, în cazul în care este ales setul 'precision, recall, f1', mai apare un câmp de completat pentru parametrul K)

3. **Graficele generate cu biblioteca Plotly**, în principal diagrame cu bare, mai puțin în pagina 'Scarcity', unde datele sunt prezentate în formatul unor diagrame cu linii.

6.2 Aspectul paginilor

Pentru a face interfața aplicației mai atractivă, mai ușor de folosit și de dezvoltat, am utilizat elemente de stilizare și componente din framework-ul Bootstrap.

Aspectul paginilor este următorul:

Figura 6.3: Pagina metricilor MAE și RMSE

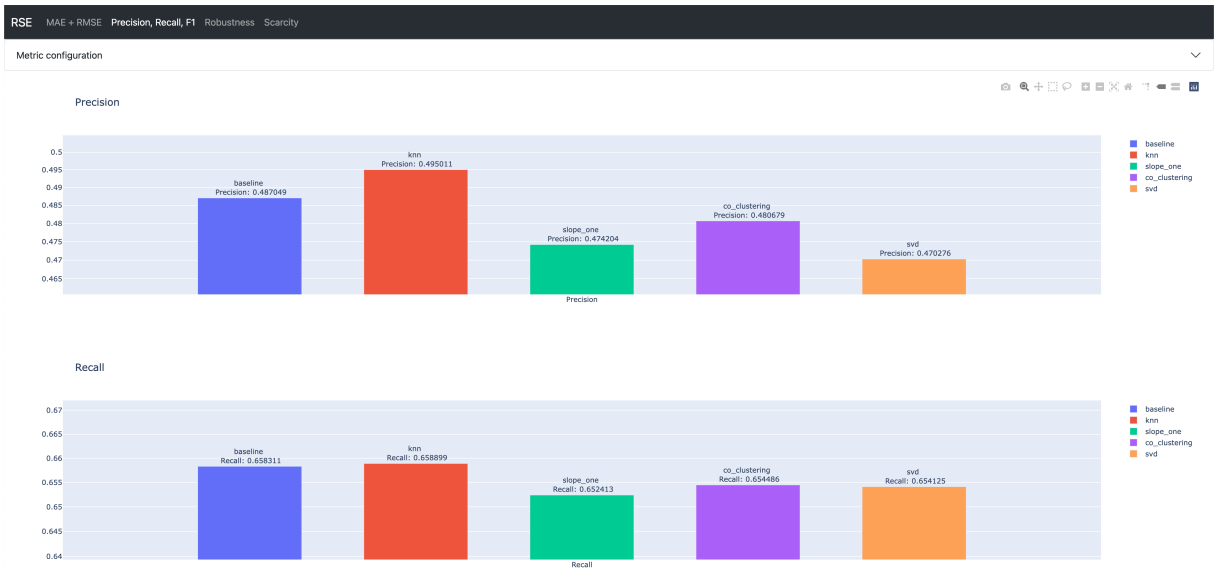

Figura 6.4: Pagina metricilor de precizie, evocare și scor F1

Figura 6.5: Pagina metricii de robustețe

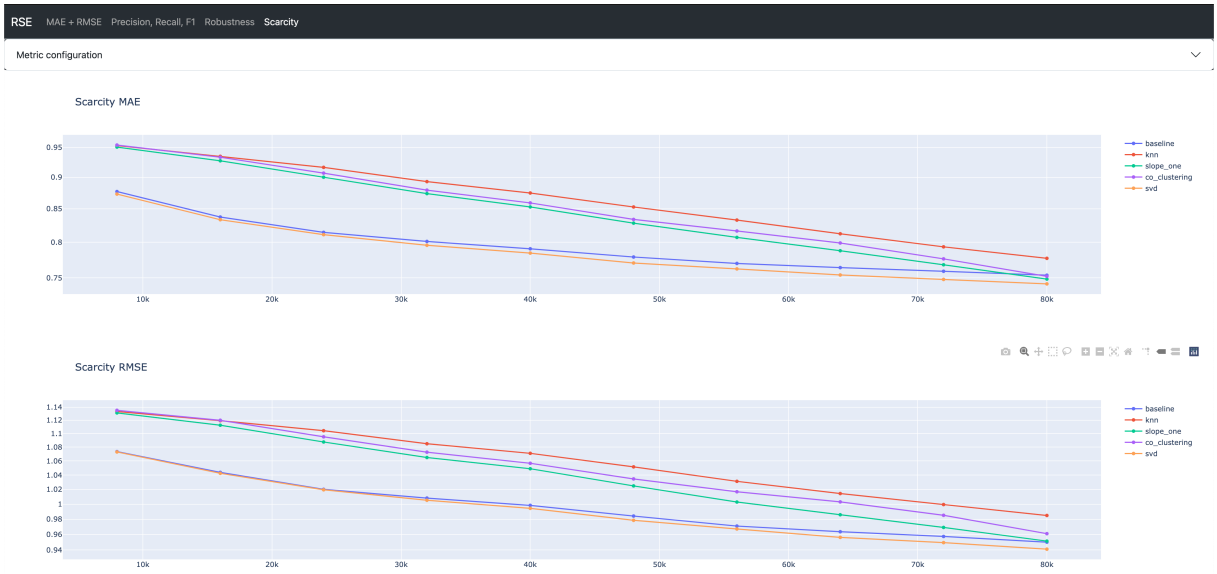

Figura 6.6: Pagina metricii deficitului de date

Capitolul 7

Concluzia

În final, am reușit să dezvolt o platformă care simplifică procesul de evaluare a performanței sistemelor de recomandare, oferind un mediu centralizat pentru colectarea, interpretarea și vizualizarea rezultatelor.

Un obiectiv de viitor al acestei aplicații ar putea fi utilizarea unui set de date introdus de utilizator, astfel evaluarea modelelor fiind personalizată pe situația utilizatorului iar rezultatele obținute vor putea fi utilizate pentru luarea deciziei corecte în legătura cu modelul potrivit. O altă opțiune ar fi posibilitatea încărcării unui fișier cu predicții, în ideea în care utilizatorul a dezvoltat un model de recomandare propriu și dorește evaluarea acestuia.

Un alt obiectiv de viitor ar putea fi integrarea a mai multor metrici de evaluare, dezvoltând în continuare această aplicație.

Pe parcursul dezvoltării acestei aplicații am învățat despre tipurile de sisteme de recomandare și modul de funcționare al acestora, precum și despre modurile de evaluare ale acestora, pentru a le putea identifica limitările sau punctele tari. Cât despre dezvoltarea de aplicații web, am învățat cum aceasta poate fi realizată folosind limbajul Python, cum pot fi integrate alte librării în aplicație, precum Plotly sau framework-ul Bootstrap, și cum se poate face comunicarea între două servere folosind cereri de tip HTTP, conform arhitecturii RESTful[15].

Bibliografie

[1] *Bootstrap*, https://getbootstrap.com/, Accesat 10 August, 2023.

[2] Thomas George și Srujana Merugu, „A scalable collaborative filtering framework based on co-clustering", în *Fifth IEEE International Conference on Data Mining (ICDM'05)*, IEEE, 2005, 4–pp.

[3] Miguel Grinberg, *Flask web development: developing web applications with python*, " O'Reilly Media, Inc.", 2018.

[4] *Gunicorn*, https://gunicorn.org/, Accesat 11 August, 2023.

[5] F Maxwell Harper și Joseph A Konstan, „The movielens datasets: History and context", în *Acm transactions on interactive intelligent systems (tiis)* 5.4 (2015), pp. 1–19.

[6] Nicolas Hug, „Surprise: A Python library for recommender systems", în *Journal of Open Source Software* 5.52 (2020), p. 2174, DOI: 10.21105/joss.02174, URL: https://doi.org/10.21105/joss.02174.

[7] Plotly Technologies Inc., *Collaborative data science*, 2015, URL: https://plot.ly.

[8] F.O. Isinkaye, Y.O. Folajimi și B.A. Ojokoh, „Recommendation systems: Principles, methods and evaluation", în *Egyptian Informatics Journal* 16.3 (2015), pp. 261–273, ISSN: 1110-8665, DOI: https://doi.org/10.1016/j.eij.2015.06.005, URL: https://www.sciencedirect.com/science/article/pii/S1110866515000341.

[9] Nima Joorabloo, Mahdi Jalili și Yongli Ren, „Improved recommender systems by denoising ratings in highly sparse datasets through individual rating confidence", în *Information Sciences* 601 (2022), pp. 242–254, ISSN: 0020-0255, DOI: https://doi.org/10.1016/j.ins.2022.03.068, URL: https://www.sciencedirect.com/science/article/pii/S0020025522002870.

[10] Hyeyoung Ko, Suyeon Lee, Yoonseo Park și Anna Choi, „A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields", în *Electronics* 11.1 (2022), ISSN: 2079-9292, DOI: 10.3390/electronics11010141, URL: https://www.mdpi.com/2079-9292/11/1/141.

[11] Yehuda Koren, „Factor in the neighbors: Scalable and accurate collaborative filtering", în *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4.1 (2010), pp. 1–24.

[12] Daniel Lemire și Anna Maclachlan, „Slope one predictors for online rating-based collaborative filtering", în *Proceedings of the 2005 SIAM International Conference on Data Mining*, SIAM, 2005, pp. 471–475, DOI: 10.1137/1.9781611972757.43, URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972757.43.

[13] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang și Guangquan Zhang, „Recommender system application developments: A survey", în *Decision Support Systems* 74 (2015), pp. 12–32, ISSN: 0167-9236, DOI: https://doi.org/10.1016/j.dss.2015.03.008, URL: https://www.sciencedirect.com/science/article/pii/S0167923615000627.

[14] Wes McKinney, „Data Structures for Statistical Computing in Python", în *Proceedings of the 9th Python in Science Conference*, ed. de Stéfan van der Walt și Jarrod Millman, 2010, pp. 56–61, DOI: 10.25080/Majora-92bf1922-00a.

[15] *What is REST API (RESTful API)*, https://www.techtarget.com/searchapparchitecture/definition/RESTful-API, Accesat 1 August, 2023.

(Acest document se leagă împreună cu lucrarea de licență/diplomă/disertație)

Declarație

Subsemnatul/Subsemnata **DIYAR MERT-DANIEL**, candidat la examenul de **licenţă/diplomă/disertaţie**, sesiunea SEPTEMBRIE 2023, la **Facultatea de Matematică şi Informatică**, programul de studii Tehnologia informaţiei, declar pe propria răspundere că lucrarea de faţă este rezultatul muncii mele, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate şi indicate, conform normelor etice, în note şi în bibliografie. Declar că nu am folosit în mod tacit sau ilegal munca altora şi că nici o parte din teză nu încalcă drepturile de proprietate intelectuală ale altcuiva, persoană fizică sau juridică. Declar că lucrarea nu a mai fost prezentată sub această formă vreunei instituţii de învăţământ superior în vederea obţinerii unui grad sau titlu ştiinţific ori didactic.

Data	Semnătura
02.09.2023	_____