

«Σχολή Θετικών Επιστημών και Τεχνολογίας»

«Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα»

Μεταπτυχιακή Διπλωματική Εργασία

«Ανάπτυξη Συστήματος Προτάσεων Ταινιών με Χρήση Μηχανικής
Μάθησης»

Καρακούλας Παναγιώτης

Επιβλέπων Καθηγητής Α΄: Καναβός Ανδρέας

Επιβλέπων Καθηγητής Β΄: Μαυρομμάτης Γεώργιος

Πάτρα, Μάιος 2026

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του/της φοιτητή/φοιτήτριας («συγγραφέας/δημιουργός») που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο/η συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του/της συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του/της συγγραφέα/δημιουργού. Ο/Η συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.



«Ανάπτυξη Συστήματος Προτάσεων Ταινιών με Χρήση Μηχανικής Μάθησης»

Καρακούλας Παναγιώτης

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής :

Καναβός Ανδρέας

Αναπληρωτής Καθηγητής
Ιόνιο Πανεπιστήμιο

Συν-Επιβλέπων Καθηγητής :

Μαυρομμάτης Γεώργιος

Μέλος ΣΕΠ ΕΑΠ

Πάτρα, Μάιος 2026

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες σε όλους όσοι συνέβαλαν στην εκπόνησή της.

Θα ήθελα να ευχαριστήσω ιδιαιτέρως τον επιβλέποντα καθηγητή μου, κ. Καναβό Ανδρέα, για την αμέριστη προσοχή, τον χρόνο και την καθοδήγηση που προσέφερε κατά τη διάρκεια της εκπόνησης της εργασίας, καθώς και για την κατανόηση και τον επαγγελματισμό που τον χαρακτηρίζουν. Επιπλέον, ευχαριστώ τον κ. Μαυρομμάτη Γεώργιο για τη συμβολή του ως συν-επιβλέποντα.

Ευχαριστώ θερμά την οικογένειά μου για τη συνεχή στήριξη, ενθάρρυνση, αγάπη και συμπαράσταση που μου προσέφεραν καθ' όλη τη διάρκεια των σπουδών μου.

Τέλος, αφιερώνω την εργασία αυτή στη μητέρα μου, Άννα, η οποία δεν έπαψε ποτέ να πιστεύει σε εμένα, και στον αδερφό μου, Πέτρο, που υπήρξε και παραμένει το στήριγμά μου σε όλη τη διάρκεια της ζωής μου.

Περίληψη

Ένα σύστημα προτάσεων είναι μια εφαρμογή που χρησιμοποιεί διάφορα δεδομένα ώστε να προβλέψει, και να παρέχει κάποιες στοχευμένες επιλογές, και να βοηθήσει στην εύρεση αυτού που αναζητούν οι άνθρωποι ανάμεσα σε έναν αυξανόμενο αριθμό επιλογών. Στην περίπτωση μας αναπτύσσουμε και υλοποιούμε ένα σύστημα προτάσεων ταινιών. Με την χρήση της βάσης δεδομένων Movielens 32M, που αναπτύχθηκε από την ερευνητική ομάδα GroupLens του Πανεπιστημίου της Μινεσότα, και την κατάλληλη επεξεργασία και καθαρισμό των δεδομένων της καταφέραμε να δημιουργήσουμε 3 μοντέλα προτάσεων ταινιών. Το πρώτο αφορά την μέθοδο Content-based filtering, η οποία προτείνει ταινίες με βάση αυτές που έχει ήδη δει ο χρήστης και τα είδη ταινιών στα οποία ανήκουν. Το δεύτερο αφορά την μέθοδο Collaborative filtering, η οποία προτείνει ταινίες με βάση την ανάλυση της συμπεριφοράς και των προτιμήσεων μιας ομάδας ανθρώπων με παρόμοιες προτιμήσεις. Τέλος, αναπτύξαμε και ένα μοντέλο με την μέθοδο Hybrid filtering, η οποία χρησιμοποιεί έναν συνδυασμό των δύο παραπάνω μεθόδων για να προτείνει ταινίες. Έπειτα ακολουθεί η αξιολόγηση των τριών αυτών μοντέλων και η εξαγωγή συμπερασμάτων.

Λέξεις – Κλειδιά

μηχανική μάθηση, σύστημα προτάσεων, σύστημα προτάσεων ταινιών, βάση δεδομένων, σύστημα προβλέψεων, φιλτράρισμα με βάση το περιεχόμενο, συνεργατικό φιλτράρισμα, υβριδικό φιλτράρισμα.

Movie Recommendation System Development through the use of Machine Learning techniques.

Panagiotis Karakoulas

Abstract

A recommendation system is an application that uses various data to predict, and provide some targeted options, and help in finding what people are looking for among a growing number of options. In our case, we develop and implement a movie recommendation system. By using the Movielens 32M database, developed by the GroupLens research group at the University of Minnesota, and by appropriately processing and cleaning its data, we managed to create 3 movie recommendation models. The first relates to the Content-based filtering method, which recommends movies based on those that the user has already seen and the genres of movies they belong to. The second relates to the Collaborative filtering method, which recommends movies based on the analysis of the behavior and preferences of a group of people with similar preferences. Last but not least, we developed a model with the Hybrid filtering method, which uses a combination of the two above methods to recommend movies. Our next step is the evaluation of these three models and the drawing of conclusions.

Keywords

recommender system, recommendation system, content-based filtering, collaborative filtering, hybrid filtering, machine learning, database, dataset

Περιεχόμενα

Περίληψη	5
Abstract	6
Περιεχόμενα	7
Κατάλογος Εικόνων / Σχημάτων	8
Κατάλογος Πινάκων	10
Συντομογραφίες & Ακρωνύμια	11
1. Συστήματα Προτάσεων (Recommender Systems)	12
1.1 Ορισμός και στόχοι	13
1.2 Χρήσεις Συστημάτων Προτάσεων	14
2. Ερευνητική Ομάδα GroupLens	15
2.1 Βάση Δεδομένων MovieLens	15
2.1.1 Κατέβασμα και Αποσυμπίεση του Dataset	16
2.1.2 Ανάλυση περιεχομένου της Βάσης Δεδομένων MovieLens 32M	17
3. Μηχανική Μάθηση	19
3.1 Εξερεύνηση Δεδομένων (EDA)	21
3.1.1 Ανάλυση αρχείου eda.py	22
3.1.2 Παρουσίαση αποτελεσμάτων αρχείου eda.py	28
3.2 Προεπεξεργασία και Ανάλυση Δεδομένων	34
3.2.1 Ανάλυση αρχείου preprocess.py	36
3.3 Μετρικές Αξιολόγησης	39
4. Μοντέλα Συστήματος Προτάσεων	45
4.1 Content-based Filtering	47
4.1.1 Εμπλουτισμός Δεδομένων μέσω TMDb	49
4.1.2 Ανάλυση αρχείου content-based.py	60
4.1.3 Παρουσίαση αποτελεσμάτων αρχείου content-based.py	88
4.1.4 Αξιολόγηση Μεθόδου	93
4.2 Collaborative Filtering	96
4.2.1 Ανάλυση αρχείου collaborative.py	98
4.2.2 Παρουσίαση αποτελεσμάτων αρχείου collaborative.py	116
4.2.3 Αξιολόγηση Μεθόδου	119
4.3 Hybrid Filtering	122
4.3.1 Ανάλυση αρχείου hybrid.py	124
4.3.2 Παρουσίαση αποτελεσμάτων αρχείου hybrid.py	158
4.3.3 Αξιολόγηση Μεθόδου	161
5. Συμπεράσματα	164
5.1 Σύγκριση των τριών μοντέλων	165
5.2 Γενικά Συμπεράσματα και Μελλοντικές Επεκτάσεις	168
Βιβλιογραφία	171

Κατάλογος Εικόνων / Σχημάτων

Εικόνα 1: Κώδικας για Κατέβασμα της Βάσης Δεδομένων	16
Εικόνα 2: Αποτελέσματα κώδικα	16
Εικόνα 3: Διαδικασίες EDA	21
Εικόνα 4: Αρχική διερευνητική ανάλυση των δεδομένων	22
Εικόνα 5: Έλεγχος ελλείψεων, διπλοτύπων και αραιότητας δεδομένων.....	23
Εικόνα 6: Κατανομή βαθμολογιών και οι 20 πιο πολυαξιολογημένες ταινίες.....	24
Εικόνα 7: Κατανομή βαθμολογιών ανά χρήστη και ανά ταινία	25
Εικόνα 8: Heatmap πίνακα χρηστών-ταινιών.....	27
Εικόνα 9: Βασικά στατιστικά της βάσης δεδομένων.....	28
Εικόνα 10: Έλεγχος ελλείψεων, διπλοτύπων και αραιότητας	30
Εικόνα 11: Διάγραμμα Κατανομής Αξιολογήσεων	31
Εικόνα 12: Διάγραμμα των 20 πιο Πολυαξιολογημένων Ταινιών	31
Εικόνα 13: Διάγραμμα Αριθμού Αξιολογήσεων ανά Χρήστη	32
Εικόνα 14: Διάγραμμα Αριθμού Αξιολογήσεων ανά Ταινία.....	32
Εικόνα 15: Διάγραμμα Αραιότητας Αλληλεπιδράσεων Χρηστών - Ταινιών.....	33
Εικόνα 16: Διαδικασίες Προεπεξεργασίας	34
Εικόνα 17: Φόρτωση και συνένωση των δεδομένων	36
Εικόνα 18: Φιλτράρισμα, κωδικοποίηση και διαχωρισμός δεδομένων.....	37
Εικόνα 19: Αποτελέσματα Προεπεξεργασίας Δεδομένων	38
Εικόνα 20: Κατηγορίες Μετρικών Αξιολόγησης	39
Εικόνα 21: Προσεγγίσεις Συστημάτων Προτάσεων.....	45
Εικόνα 22: Content Based Filtering.....	47
Εικόνα 23: Ρύθμιση διαδρομών και σύνδεσης με το TMDB API.....	50
Εικόνα 24: Ανάκτηση μεταδεδομένων από το TMDB API	51
Εικόνα 25: Εξαγωγή μεταδεδομένων ταινιών από το TMDB	52
Εικόνα 26: Έλεγχος αρχείων και επιλογή ταινιών για TMDB εμπλουτισμό	53
Εικόνα 27: Φιλτράρισμα και έλεγχος ταινιών προς εμπλουτισμό.....	54
Εικόνα 28: Δημιουργία εμπλουτισμένων μεταδεδομένων ταινίας	56
Εικόνα 29: Αποθήκευση εμπλουτισμένων δεδομένων TMDB	58
Εικόνα 30: Ρυθμίσεις και διαδρομές του συστήματος συστάσεων	60
Εικόνα 31: Καθαρισμός κειμένου και υπολογισμός βαρών αξιολογήσεων	62
Εικόνα 32: Υλοποίηση μετρικών αξιολόγησης για τις προτεινόμενες ταινίες.....	64
Εικόνα 33: Υπολογισμός ποικιλομορφίας στις προτεινόμενες ταινίες.....	66
Εικόνα 34: Δημιουργία καταλόγου υποψήφιων ταινιών	68
Εικόνα 35: Καθαρισμός ειδών και ετικετών ταινιών.....	70
Εικόνα 36: Ενσωμάτωση μεταδεδομένων TMDB.....	72
Εικόνα 37: Διαχείριση ελλিপών τιμών στα μεταδεδομένα TMDB.....	73
Εικόνα 38: Δημιουργία εμπλουτισμένου κειμενικού περιεχομένου για κάθε ταινία	74
Εικόνα 39: Δημιουργία TF-IDF πίνακα και υπολογισμός δημοφιλίας ταινιών	76
Εικόνα 40: Δημιουργία προφίλ χρήστη	78
Εικόνα 41: Παραγωγή συστάσεων ανά χρήστη.....	80
Εικόνα 42: Αξιολόγηση συστήματος συστάσεων	82

Εικόνα 43: Υπολογισμός μετρικών αξιολόγησης.....	83
Εικόνα 44: Ιστορικό θετικών αξιολογήσεων χρήστη	85
Εικόνα 45: Εκτέλεση και αξιολόγηση του μοντέλου συστάσεων	86
Εικόνα 46: Αποτελέσματα προεπεξεργασίας και TF-IDF αναπαράστασης.....	88
Εικόνα 47: Θετικά αξιολογημένες ταινίες χρήστη	90
Εικόνα 48: Συστάσεις και αξιολόγηση content-based μοντέλου.....	92
Εικόνα 49: Collaborative Filtering	96
Εικόνα 50: Ρυθμίσεις collaborative filtering μοντέλου	98
Εικόνα 51: Μετρικές αξιολόγησης collaborative filtering	100
Εικόνα 52: Ποικιλομορφία συστάσεων collaborative filtering	102
Εικόνα 53: Δημιουργία καταλόγου υποψήφιων ταινιών στο collaborative filtering.....	104
Εικόνα 54: Δημιουργία πίνακα αλληλεπιδράσεων	106
Εικόνα 55: Εκπαίδευση και αποθήκευση ALS μοντέλου.....	108
Εικόνα 56: Παραγωγή Top-N συστάσεων με ALS	109
Εικόνα 57: Αξιολόγηση ALS collaborative filtering μοντέλου.....	111
Εικόνα 58: Υπολογισμός μετρικών αξιολόγησης ALS	113
Εικόνα 59: Εκτέλεση και αξιολόγηση collaborative filtering μοντέλου	114
Εικόνα 60: Αποτελέσματα προεπεξεργασίας ALS μοντέλου.....	116
Εικόνα 61: Top-10 συστάσεις ALS για χρήστη	117
Εικόνα 62: Αποτελέσματα αξιολόγησης collaborative filtering.....	118
Εικόνα 63: Hybrid Filtering.....	122
Εικόνα 64: Ρυθμίσεις hybrid συστήματος συστάσεων	124
Εικόνα 65: Βοηθητικές συναρτήσεις του hybrid συστήματος.....	127
Εικόνα 66: Κανονικοποίηση και μετρικές αξιολόγησης hybrid μοντέλου.....	129
Εικόνα 67: Μετρικές κατάταξης και ποικιλομορφίας hybrid μοντέλου.....	131
Εικόνα 68: Δημιουργία καταλόγου υποψήφιων ταινιών στο hybrid μοντέλο.....	133
Εικόνα 69: Καθαρισμός ταινιών και tags στο hybrid μοντέλο.....	135
Εικόνα 70: Ενσωμάτωση μεταδεδομένων TMDb στο hybrid μοντέλο	137
Εικόνα 71: Δημιουργία rich content στο hybrid μοντέλο.....	139
Εικόνα 72: TF-IDF αναπαράσταση και αντιστοίχιση IDs στο hybrid μοντέλο	141
Εικόνα 73: Πίνακας αλληλεπιδράσεων και εκπαίδευση ALS στο hybrid μοντέλο.....	143
Εικόνα 74: Δημιουργία προφίλ χρήστη στο hybrid μοντέλο.....	146
Εικόνα 75: Αρχικοποίηση υβριδικών συστάσεων	148
Εικόνα 76: Συνδυασμός scores στο hybrid μοντέλο.....	150
Εικόνα 77: Αξιολόγηση hybrid συστήματος συστάσεων	152
Εικόνα 78: Υπολογισμός μετρικών αξιολόγησης hybrid μοντέλου	154
Εικόνα 79: Εκτέλεση και αξιολόγηση hybrid μοντέλου.....	156
Εικόνα 80: Προεπεξεργασία και ιστορικό χρήστη στο hybrid μοντέλο.....	158
Εικόνα 81: Συστάσεις και αξιολόγηση hybrid μοντέλου.....	160

Κατάλογος Πινάκων

Πίνακας 1: Σύγκριση μετρήσεων των 3 μοντέλων	165
--	-----

Συντομογραφίες & Ακρωνύμια

ΕΑΠ	Ελληνικό Ανοικτό Πανεπιστήμιο
ALS	Alternating Least Squares
CF	Collaborative Filtering — Συνεργατικό φιλτράρισμα
CSV	Comma-Separated Values — Αρχείο τιμών διαχωρισμένων με κόμμα
EDA	Exploratory Data Analysis — Εξερευνητική Ανάλυση Δεδομένων
ID	Identifier — Αναγνωριστικό
MAP	Mean Average Precision
NDCG	Normalized Discounted Cumulative Gain
TF-IDF	Term Frequency–Inverse Document Frequency
TMDb	The Movie Database
Top-K	Οι K κορυφαίες προτάσεις/επιλογές
HitRate	Ποσοστό επιτυχίας προτάσεων

1. Συστήματα Προτάσεων (Recommender Systems)

Στην σημερινή εποχή, με την τεχνολογία να εξελίσσεται ραγδαία και με τον κλάδο των διαδικτυακών υπηρεσιών σε άνθηση, ο άνθρωπος εκτίθεται καθημερινά σε ένα ολοένα και αυξανόμενο όγκο πληροφοριών (Roy & Dutta, 2022). Παρότι αυτή η πληθώρα πληροφοριών ενισχύει τις δυνατότητες ενημέρωσης, ταυτόχρονα προκύπτει και το πρόβλημα της πληροφοριακής συμφόρησης. Καθημερινά παράγεται ένας τεράστιος όγκος δεδομένων από κριτικές, σχόλια και αξιολογήσεις προϊόντων, δυσκολεύοντας την ανίχνευση περιεχομένου που ανταποκρίνεται στις πραγματικές ανάγκες του χρήστη. Κατά αυτό τον τρόπο, τα συστήματα προτάσεων αποτελούν ένα κρίσιμο εργαλείο για την καθοδήγηση των χρηστών μέσα σε αυτόν τον κυκεώνα δεδομένων.

Η πληροφοριακή συμφόρηση δεν αφορά μόνο την ποσότητα των διαθέσιμων δεδομένων, αλλά και τη δυσκολία του χρήστη να αξιολογήσει ποια από αυτά είναι πραγματικά χρήσιμα για τις προσωπικές του ανάγκες. Σε πολλές περιπτώσεις, ο χρήστης δεν έχει τον χρόνο ή τη δυνατότητα να εξετάσει όλες τις διαθέσιμες επιλογές, με αποτέλεσμα η διαδικασία επιλογής να γίνεται πιο σύνθετη και λιγότερο αποδοτική. Τα συστήματα προτάσεων έρχονται να αντιμετωπίσουν αυτό το πρόβλημα, λειτουργώντας ως μηχανισμοί φιλτραρίσματος και υποστήριξης αποφάσεων. Μέσα από την ανάλυση δεδομένων που σχετίζονται με τις προτιμήσεις, τη συμπεριφορά ή τις προηγούμενες επιλογές των χρηστών, μπορούν να περιορίσουν τον μεγάλο όγκο διαθέσιμων αντικειμένων και να αναδείξουν εκείνα που είναι πιθανότερο να ενδιαφέρουν τον εκάστοτε χρήστη. Έτσι, η αναζήτηση γίνεται πιο στοχευμένη, ενώ η εμπειρία χρήσης γίνεται πιο άμεση, προσωποποιημένη και αποτελεσματική (Roy & Dutta, 2022).

Η χρήση συστημάτων προτάσεων έχει επεκταθεί, πλέον, σημαντικά και αποτελεί αναπόσπαστο συστατικό τόσο σε ψυχαγωγικές εφαρμογές (όπως ταινίες, μουσική και τηλεοπτικό περιεχόμενο) όσο και σε τομείς όπως το ηλεκτρονικό εμπόριο, η τουριστική βιομηχανία, η εκπαίδευση και οι επιστημονικές εφαρμογές. Η ανάγκη για συστήματα που έχουν την δυνατότητα να προσαρμόζονται στις προτιμήσεις του χρήστη/καταναλωτή και να προσφέρουν εξατομικευμένες προτάσεις υψηλής ποιότητας μοιάζει πιο επιτακτική από ποτέ (Roy & Dutta, 2022).

1.1 Ορισμός και στόχοι

Τα Συστήματα Προτάσεων είναι εργαλεία λογισμικού, τα οποία χρησιμοποιούν συγκεκριμένες τεχνικές για να προτείνουν σε κάποιους χρήστες υπηρεσίες, αντικείμενα και προϊόντα τα οποία θα είναι της αρεσκείας τους και πιθανόν να τους φανούν χρήσιμα. Κάθε σύστημα προτάσεων είναι σχεδιασμένο με επίκεντρο την εκάστοτε συγκεκριμένη κατηγορία αντικειμένων ενώ το περιβάλλον χρήσης και οι αλγοριθμικές διαδικασίες του προσαρμόζονται ανάλογα ώστε να παρέχουν πιο αξιόπιστες και χρήσιμες συστάσεις. Σε θεμελιώδες επίπεδο, τα συστήματα προτάσεων βασίζονται στην αλληλεπίδραση μεταξύ δύο κεντρικών οντοτήτων: των χρηστών και των αντικειμένων/υπηρεσιών. Κάθε χρήστης εκφράζει κάποια μορφή προτίμησης προς ορισμένα αντικείμενα, είτε άμεσα, μέσω αξιολογήσεων, είτε έμμεσα, μέσω της συμπεριφοράς του κατά την χρήση του λογισμικού. Η βασική πρόκληση για κάθε σύστημα προτάσεων είναι η εκτίμηση της πιθανής προτίμησης ενός χρήστη για αντικείμενα που δεν έχει αξιολογήσει και, γενικευμένα, χρησιμοποιήσει. Η απόδοση ενός τέτοιου συστήματος επηρεάζεται έντονα από τον τρόπο που αξιοποιεί τα διαθέσιμα δεδομένα, τα οποία μπορεί να προκύπτουν σε διάφορες μορφές, και από τον αλγόριθμο που εφαρμόζεται για την παραγωγή των τελικών προτάσεων (“Recommender Systems Handbook,” 2011; Roy & Dutta, 2022).

Στο πλαίσιο αυτό, γίνεται φανερό ότι τα συστήματα προτάσεων δεν λειτουργούν απλώς ως μηχανισμοί εμφάνισης δημοφιλών επιλογών, αλλά ως εργαλεία εξατομίκευσης της πληροφορίας. Ο βασικός τους ρόλος είναι να περιορίζουν τον μεγάλο αριθμό διαθέσιμων επιλογών και να παρουσιάζουν στον χρήστη εκείνες που εμφανίζουν μεγαλύτερη πιθανότητα να ανταποκρίνονται στα ενδιαφέροντά του. Με αυτόν τον τρόπο, συμβάλλουν τόσο στη βελτίωση της εμπειρίας χρήσης όσο και στην αποτελεσματικότερη αξιοποίηση των δεδομένων που παράγονται από τις αλληλεπιδράσεις των χρηστών με το εκάστοτε σύστημα (Adomavicius & Tuzhilin, 2005).

1.2 Χρήσεις Συστημάτων Προτάσεων

Ως ερευνητικό πεδίο τα συστήματα προτάσεων εμφανίστηκαν αυτόνομα την δεκαετία το 1990 και γνώρισαν ραγδαία ανάπτυξη στη συνέχεια. Η σπουδαιότητα τους επιβεβαιώνεται και από την υιοθέτηση τους από κορυφαίες στο είδος τους πλατφόρμες, όπως η Amazon, το Netflix, το YouTube, η TripAdvisor, η Yahoo, το IMDB κ.α. Μάλιστα, κατά το παρελθόν, η Netflix είχε θεσπίσει διαγωνισμό για την βελτίωση της ακρίβειας του συστήματός της, προσφέροντας έπαθλο της τάξεως του ενός εκατομμυρίου δολαρίων. Ποιοι είναι όμως, οι πραγματικοί λόγοι για την υιοθέτηση ενός συστήματος προτάσεων;

Λόγος εξαιρετικής σημασίας είναι η αύξηση των πωλήσεων. Η βασικότερη λειτουργία ενός συστήματος προτάσεων είναι η προώθηση προϊόντων, πέρα από αυτά που θα αγόραζε ο χρήστης χωρίς την ύπαρξη προτάσεων, διότι τα προτεινόμενα προϊόντα είναι πολύ πιθανό να ταιριάζουν στις ανάγκες και προτιμήσεις του χρήστη. Στόχος είναι η αύξηση της μετατροπής. Από την πρόταση, να προκύψει πραγματική κατανάλωση.

Μία ακόμη αφορμή υιοθέτησης συστήματος προτάσεων είναι η προώθηση μιας ποικιλίας προϊόντων, που υπό διαφορετικές συνθήκες θα ήταν δύσκολο να εντοπιστούν. Πιο συγκεκριμένα, η προβολή λιγότερο δημοφιλών αντικειμένων σε κάποιο χρήστη, θα ήταν επικίνδυνη χωρίς την διασταύρωση με τις προτιμήσεις αυτού του συγκεκριμένου χρήστη.

Σημαντική, επίσης, είναι και η βελτίωση της ικανοποίησης των χρηστών. Ένα καλοσχεδιασμένο και εύχρηστο σύστημα προτάσεων ενισχύει την συνολική εμπειρία του καταναλωτή, παρέχοντας του σχετικές και ενδιαφέρουσες προτάσεις. Έτσι, ενισχύεται η χρήση και η αποδοχή των προτάσεων του συστήματος. Υπό αυτό το πρίσμα, βελτιώνεται και η πιστότητα των χρηστών. Οι χρήστες συνεχίζουν να αλληλοεπιδρούν και οι συστάσεις ολοένα και βελτιώνονται και γίνονται πιο προσαρμοσμένες (“Recommender Systems Handbook,” 2011).

Επιπλέον, η αξία των συστημάτων προτάσεων δεν περιορίζεται μόνο στην εμπορική αξιοποίησή τους, αλλά συνδέεται και με την καλύτερη οργάνωση και αξιοποίηση της πληροφορίας. Σε περιβάλλοντα όπου οι διαθέσιμες επιλογές είναι πάρα πολλές, ο χρήστης συχνά δυσκολεύεται να εντοπίσει γρήγορα αυτό που πραγματικά τον ενδιαφέρει. Ένα σύστημα προτάσεων μπορεί να λειτουργήσει ως ενδιάμεσος μηχανισμός ανάμεσα στον χρήστη και τον μεγάλο όγκο διαθέσιμου περιεχομένου, περιορίζοντας τις επιλογές και παρουσιάζοντας εκείνες που θεωρούνται πιο σχετικές.

2. Ερευνητική ομάδα GroupLens

Η GroupLens αποτελεί ερευνητική ομάδα του Τμήματος Επιστήμης Υπολογιστών και Μηχανικής του Πανεπιστημίου της Minnesota. Η συγκεκριμένη ομάδα ασχολείται με τομείς όπως οι ψηφιακές βιβλιοθήκες, τα γεωγραφικά πληροφοριακά συστήματα, τα συστήματα προτάσεων, οι τεχνολογίες και, γενικότερα, οι διαδικτυακές κοινότητες. Η αποστολή της ομάδας, όπως την διατυπώνει και η ίδια, είναι να εξελίξει την επιστήμη της κοινωνικής υπολογιστικής, δημιουργώντας και μελετώντας τεχνολογίες που χρησιμοποιούνται από πραγματικούς χρήστες, με στόχο την προώθηση τόσο της θεωρητικής γνώσης όσο και της πρακτικής εφαρμογής. Υπό αυτό το πρίσμα δημιουργήθηκε από την GroupLens, η βάση δεδομένων MovieLens (F. Maxwell Harper and Joseph A. Konstan. 2015).

2.1 Βάση δεδομένων MovieLens

Η ερευνητική ομάδα GroupLens έχει συγκεντρώσει και δημοσιοποιήσει σύνολα δεδομένων με αξιολογήσεις χρηστών, τα οποία προέρχονται από την πλατφόρμα MovieLens. Τα προαναφερθέντα δεδομένα έχουν συλλεχθεί σε διάφορες χρονικές περιόδους, εξ 'ου και η ποικιλομορφία στον όγκο τους. Στην παρούσα εργασία, για να εξυπηρετήσουμε τους ερευνητικούς μας σκοπούς, επιλέξαμε το MovieLens 32M. Το συγκεκριμένο σύνολο δεδομένων αποτυπώνει την αλληλεπίδραση των χρηστών με την υπηρεσία προτάσεων ταινιών της πλατφόρμας MovieLens, τόσο μέσω βαθμολογιών σε κλίμακες 0.5 – 5 αστεριών, όσο και μέσω ελεύθερης απόδοσης ετικετών ανά ταινία (tags). Στην ολότητα της περιλαμβάνει 32.000.204 αξιολογήσεις και 2.000.072 ετικέτες οι οποίες αντιστοιχούν σε 87.585 ταινίες. Οι καταχωρήσεις προέρχονται από 200.948 χρήστες και συλλέχθηκαν σε ένα διάστημα από Ιανουάριο του 1995 έως Οκτώβριο του 2023, όταν και δημοσιεύθηκαν. Οι χρήστες επιλέχθηκαν τυχαία, με μοναδική προϋπόθεση να έχουν αξιολογήσει τουλάχιστον 20 ταινίες (F. Maxwell Harper and Joseph A. Konstan. 2015).

2.1.1 Κατέβασμα και Αποσυμπίεση της Βάσης Δεδομένων MovieLens

Ο παρακάτω κώδικας είναι υπεύθυνος για την λήψη του MovieLens 32M dataset από τον επίσημο ιστότοπο του GroupLens, καθώς και για την αποσυμπίεση και αποθήκευση των αρχείων στην καθορισμένη διαδρομή του έργου μας.

```
data_download.py ×
1  import os
2  import urllib.request
3  import zipfile
4
5
6  dataPath = os.path.join(os.path.dirname(__file__), '..', 'data')
7  #ο φάκελος υπάρχει ήδη, οπότε η γραμμή διατηρείται για λόγους φορητότητας
8  os.makedirs(dataPath, exist_ok=True)
9
10
11  dataUrl = 'http://files.grouplens.org/datasets/movielens/ml-32m.zip'
12  zipPath = os.path.join(dataPath, 'ml-32m.zip')
13
14
15  print('Κατέβασμα της Βάσης Δεδομένων')
16  urllib.request.urlretrieve(dataUrl, zipPath)
17  print('Unzipping...')
18  with zipfile.ZipFile(zipPath, mode: 'r') as z:
19      z.extractall(dataPath)
20
21
22  print('Η διαδικασία ολοκληρώθηκε. Περιεχόμενα φακέλου: ', os.listdir(dataPath))
```

Εικόνα 1: Κώδικας για Κατέβασμα της Βάσης Δεδομένων

```
Κατέβασμα της Βάσης Δεδομένων
Unzipping...
Η διαδικασία ολοκληρώθηκε. Περιεχόμενα φακέλου:  ['ml-32m', 'ml-32m.zip']

Process finished with exit code 0
```

Εικόνα 2: Αποτελέσματα κώδικα

Στον παραπάνω κώδικα, αρχικά εισάγουμε τις βιβλιοθήκες που μας ενδιαφέρουν:

- os: Χρησιμοποιείται για την δημιουργία φακέλων και διαδρομών.

- `urllib.request`: Επιτρέπει την λήψη αρχείων μέσω HTTP request (μηνύματα από κάποιον client σε κάποιο server για την πραγματοποίηση κάποιας ενέργειας).
- `zipfile`: Υπεύθυνη για την αποσυμπίεση αρχείων.

Έπειτα, ορίζουμε τον φάκελο που θα αποθηκεύσουμε τα δεδομένα. Μέσω της μεταβλητής `dataPath`, βρίσκουμε που υπάρχει το αρχείο μας (`data_download.py`), μετακινούμαστε έναν φάκελο πάνω και, αν ο φάκελος `data` δεν υπάρχει τον δημιουργούμε, αν υπάρχει δεν κάνουμε τίποτα. Στο επόμενο βήμα, αποθηκεύουμε την διεύθυνση της βάσης δεδομένων από όπου θα κατεβάσουμε τα αρχεία μας και ορίζουμε την διαδρομή στην οποία θα μεταφερθούν όταν τα κατεβάσουμε. Στην συνέχεια κάνουμε το αίτημα μας για κατέβασμα μέσω της μεθόδου `urlretrieve()`, ανοίγουμε το zip αρχείο σε λειτουργία ανάγνωσης και αποσυμπιέζουμε το περιεχόμενο του στο φάκελο `data`. Τέλος, Εμφανίζουμε όλα τα αρχεία που υπάρχουν πλέον στον φάκελο `data`.

2.1.2 Ανάλυση Περιεχομένου της Βάσης Δεδομένων MovieLens 32M

Η MovieLens περιλαμβάνει 4 βασικά αρχεία. Το `links.csv`, το `movies.csv`, το `ratings.csv` και το `tags.csv`. Οι αξιολογήσεις βρίσκονται στο αρχείο `ratings.csv`, με διάστημα από 0.5 έως 5.0, το οποίο έχει την εξής μορφή:

- `userId, movieId, rating, timestamp`.

Οι ετικέτες βρίσκονται στο αρχείο `tags.csv` και αποτελούν μεταδεδομένα των χρηστών σχετικά με τις ταινίες. Η μορφή του είναι η ακόλουθη:

- `userId, movieId, tag, timestamp`.

Οι πληροφορίες για τις ταινίες βρίσκονται στο αρχείο `movies.csv`, με βασικά συστατικά τον τίτλο και το είδος και έχουν την ακόλουθη μορφή:

- `movieId, title, genres`.

Τα genres επιλέγονται από τα εξής:

Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western.

Τα `userId` είναι συνεπή μεταξύ των αρχείων `ratings.csv` και `tags.csv`, όπως αντίστοιχα και τα `movieId` είναι συνεπή μεταξύ των `ratings.csv`, `tags.csv` και `movies.csv`. Επιπλέον, στην παρούσα εργασία αξιοποιείται και το αρχείο `links.csv`, το οποίο λειτουργεί ως συνδετικός πίνακας ανάμεσα στα αναγνωριστικά του MovieLens και σε εξωτερικές βάσεις δεδομένων ταινιών. Συγκεκριμένα, το αρχείο αυτό περιλαμβάνει για κάθε ταινία το `movieId`, δηλαδή το

εσωτερικό αναγνωριστικό της ταινίας στο MovieLens, καθώς και τα αντίστοιχα imdbId και tmdbId, τα οποία παραπέμπουν στις αντίστοιχες εγγραφές της ταινίας στο IMDb και στο TMDb. Επομένως, το links.csv δεν περιέχει άμεσα αξιολογήσεις ή περιγραφικά χαρακτηριστικά ταινιών, αλλά παρέχει τη δυνατότητα σύνδεσης του MovieLens dataset με εξωτερικές πηγές πληροφορίας.

Στο πλαίσιο της παρούσας εργασίας, το links.csv χρησιμοποιήθηκε για τον εμπλουτισμό του συνόλου δεδομένων με πρόσθετα μεταδεδομένα από το TMDb. Μέσω του πεδίου tmdbId, κάθε ταινία του MovieLens αντιστοιχίστηκε, όπου ήταν δυνατό, στην αντίστοιχη εγγραφή της στο TMDb API. Με τον τρόπο αυτό ανακτήθηκαν επιπλέον πληροφορίες, όπως περίληψη, tagline, λέξεις-κλειδιά, σκηνοθέτης, cast, εταιρείες παραγωγής, συλλογή, αρχικός τίτλος και είδη ταινιών σύμφωνα με το TMDb. Τα δεδομένα αυτά αποθηκεύτηκαν στο αρχείο tmdb_enriched_movies.csv και στη συνέχεια αξιοποιήθηκαν στο content-based και στο hybrid μοντέλο για τη δημιουργία πιο πλούσιας κειμενικής αναπαράστασης των ταινιών. Συνεπώς, το links.csv αποτέλεσε κρίσιμο ενδιάμεσο αρχείο, καθώς επέτρεψε τη σύνδεση των αρχικών δεδομένων του MovieLens με εξωτερική πληροφορία περιεχομένου.

Συνολικά, η ύπαρξη του links.csv ενισχύει τη δυνατότητα επέκτασης του MovieLens dataset πέρα από τις αρχικές του στήλες. Ενώ τα ratings.csv και tags.csv παρέχουν πληροφορία συμπεριφοράς και χρήστη, και το movies.csv παρέχει βασικό περιεχόμενο, το links.csv επιτρέπει την άντληση συμπληρωματικών χαρακτηριστικών από εξωτερικές βάσεις. Αυτό ήταν ιδιαίτερα σημαντικό για την παρούσα εργασία, καθώς η απλή πληροφορία των genres και των tags δεν κρίθηκε επαρκής για την πιο εκφραστική αναπαράσταση του περιεχομένου των ταινιών. Μέσω του TMDb εμπλουτισμού, το τελικό dataset απέκτησε περισσότερα περιγραφικά στοιχεία, τα οποία βοήθησαν στη δημιουργία του rich_content και, κατ' επέκταση, στη βελτίωση της πληροφορίας που χρησιμοποιήθηκε από τα content-based και hybrid συστήματα σύστασης (F. Maxwell Harper and Joseph A. Konstan. 2015).

3. Μηχανική Μάθηση

Η μηχανική μάθηση αποτελεί υποκλάδο της τεχνητής νοημοσύνης και ασχολείται με την ανάπτυξη συστημάτων που μπορούν να βελτιώνουν τη συμπεριφορά τους μέσα από την εμπειρία και την αξιοποίηση δεδομένων. Σύμφωνα με την βιβλιογραφία, η μηχανική μάθηση εξετάζει το πώς μπορούν να κατασκευαστούν υπολογιστικά συστήματα που βελτιώνονται αυτόματα μέσω της εμπειρίας (Jordan, M. I., & Mitchell, T. M., 2015), ενώ, ακόμη, επισημαίνεται ότι οι τεχνικές μηχανικής μάθησης χρησιμοποιούνται για την εξαγωγή γνώσης, την αναγνώριση μοτίβων και την υποστήριξη προβλέψεων ή αποφάσεων σε διαφορετικά πεδία εφαρμογής. Σε αντίθεση με τον παραδοσιακό προγραμματισμό, όπου οι κανόνες λειτουργίας ενός συστήματος ορίζονται ρητά από τον προγραμματιστή, στη μηχανική μάθηση το μοντέλο εκπαιδεύεται πάνω σε δεδομένα και προσπαθεί να εντοπίσει σχέσεις που μπορούν να γενικευθούν σε νέες περιπτώσεις. Για τον λόγο αυτό, η ποιότητα και η καταλληλότητα των δεδομένων αποτελούν κρίσιμο παράγοντα για την αξιοπιστία του τελικού μοντέλου (Sarker, 2021).

Πριν από την εφαρμογή οποιουδήποτε αλγορίθμου, είναι απαραίτητο να προηγηθεί διερευνητική ανάλυση δεδομένων, καθώς η EDA επιτρέπει την κατανόηση της δομής του dataset, την αναγνώριση βασικών κατανομών, την ανίχνευση ακραίων τιμών, ελλείψεων και πιθανών ασυνεπειών. Η διερευνητική ανάλυση δεδομένων αποτελεί σημαντικό στάδιο της στατιστικής και υπολογιστικής ανάλυσης, καθώς βοηθά τον ερευνητή να εξετάσει τα δεδομένα πριν από την εφαρμογή πιο τυπικών μοντέλων ή συμπερασματικών διαδικασιών. Με άλλα λόγια, η EDA λειτουργεί ως αρχικό στάδιο κατανόησης και ελέγχου των δεδομένων, επιτρέποντας να εντοπιστούν χαρακτηριστικά ή προβλήματα που μπορούν να επηρεάσουν τα επόμενα βήματα της μοντελοποίησης (Morgenthaler, 2009).

Παράλληλα, η προεπεξεργασία δεδομένων αποτελεί αναγκαίο στάδιο για τη μετατροπή των αρχικών δεδομένων σε μορφή κατάλληλη για μοντελοποίηση. Στην βιβλιογραφία, η προεπεξεργασία περιγράφεται ως το σύνολο διαδικασιών που γεφυρώνουν το χάσμα ανάμεσα στη συλλογή των δεδομένων και την εφαρμογή αλγορίθμων εξόρυξης γνώσης, περιλαμβάνοντας ενέργειες όπως καθαρισμό, μετασχηματισμό, κανονικοποίηση, μείωση θορύβου και επιλογή κατάλληλων εγγραφών ή χαρακτηριστικών (García et al., 2015). Στο πλαίσιο της παρούσας εργασίας, τα στάδια αυτά είναι ιδιαίτερα σημαντικά, καθώς το MovieLens 32M αποτελεί μεγάλο και αραιό dataset αξιολογήσεων χρηστών-ταινιών.

Επομένως, η διερευνητική ανάλυση, ο έλεγχος ποιότητας, το φιλτράρισμα αραιών δεδομένων και ο διαχωρισμός σε train και test set δεν αποτελούν απλές βοηθητικές διαδικασίες, αλλά βασικά βήματα που καθορίζουν την αξιοπιστία της μετέπειτα ανάπτυξης και αξιολόγησης των συστημάτων σύστασης.

3.1 Διερευνητική Ανάλυση Δεδομένων (EDA)

Η Διερευνητική Ανάλυση Δεδομένων είναι η διαδικασία κατανόησης των δεδομένων μας πριν υλοποιήσουμε κάποιο μοντέλο. Η EDA δεν είναι κάποιο μαθησιακό μοντέλο, αντιθέτως είναι ένα προαπαιτούμενο για να εφαρμόσουμε μηχανική μάθηση. Στο στάδιο αυτό, επιχειρούμε μια αρχική, συστηματική χαρτογράφηση του συνόλου των δεδομένων, με σκοπό την αναγνώριση μοτίβων, πιθανών θορύβων και ακραίων τιμών και τον έλεγχο προκαταρκτικών υποθέσεων μέσω βασικής στατιστικής και γραφικών αναπαραστάσεων. Η διαδικασία αυτή δεν αποτελεί απλώς ένα εισαγωγικό βήμα, αλλά έναν καθοριστικό μηχανισμό που θα τροφοδοτήσει τα επόμενα μας βήματα στην ανάλυση. Μάλιστα, πολλά από τα συμπεράσματα που προκύπτουν στην EDA επηρεάζουν άμεσα τον τρόπο με τον οποίο θα σχεδιάσουμε το μοντέλο μας (Da Poian et al., 2023).



Εικόνα 3: Διαδικασίες EDA, Πηγή: LeewayHertz

3.1.1 Ανάλυση αρχείου eda.py

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 # Φόρτωση ratings και movies
5 ratings = pd.read_csv('../data/ml-32m/ratings.csv')
6 movies = pd.read_csv('../data/ml-32m/movies.csv')
7
8 print("Φόρτωση βαθμολογιών:", ratings.shape)
9 print("Φόρτωση ταινιών:", movies.shape)
10
11 # συνένωση ratings και movies στο movieId ώστε να φαίνεται και ο τίτλος της ταινίας
12 data = ratings.merge(movies[['movieId', 'title']], on='movieId')
13 print("\nΣυνένωση δεδομένων:", data.shape)
14
15 # Βασικά στατιστικά
16 print("\nΔείγμα πρώτων γραμμών:")
17 print(data.head())
18
19 print("\nΜέση Βαθμολογία:", round(data['rating'].mean(), 2)) #με δύο δεκαδικά ψηφία
20 print("Αριθμός Μοναδικών Χρηστών:", data['userId'].nunique())
21 print("Αριθμός Μοναδικών Ταινιών:", data['movieId'].nunique())
22
23 # Ratings ανα χρήστη
24 ratings_per_user = data.groupby('userId')['rating'].count()
25 print("\nΑξιολογήσεις ανα χρήστη:")
26 print(ratings_per_user.describe()) #μέσο όρο, διάμεσο, min, max
27
28 # Ratings ανα ταινία
29 ratings_per_movie = data.groupby('movieId')['rating'].count()
30 print("\nRatings per movie : ")
31 print(ratings_per_movie.describe()) #μέσο όρο, διάμεσο, min, max
```

Εικόνα 4: Αρχική διερευνητική ανάλυση των δεδομένων

Η παρούσα εφαρμογή εστιάζει στην προεπεξεργασία και τη στατιστική ανάλυση του συνόλου δεδομένων MovieLens την οποία επιλέξαμε για την εργασία μας. Ουσιαστικός σκοπός της εφαρμογής είναι η κατανόηση της δομής και των εγγενών χαρακτηριστικών των δεδομένων πριν την εφαρμογή αλγορίθμων μηχανικής μάθησης.

Η διαδικασία ξεκινά με την εισαγωγή δύο βασικών βιβλιοθηκών, της pandas, η οποία παρέχει εργαλεία προς διευκόλυνση της επεξεργασίας, φιλτραρίσματος, ομαδοποίησης και ανάλυσης δεδομένων, και της matplotlib. pyplot, η οποία χρησιμοποιείται για την δημιουργία γραφημάτων, επιτρέποντας μας έτσι την οπτικοποίηση διαφόρων δεδομένων που

προκύπτουν από την βάση μας. Έπειτα, φορτώνουμε τα πρωτογενή δεδομένα των ratings και monies και εκτελούμε μια εκτύπωση των γραμμών και στηλών λειτουργώντας ως το πρώτο βήμα επιβεβαίωσης της ορθότητας των δεδομένων. Ακολουθεί η φάση της συνένωσης των δύο πινάκων βάσει του αναγνωριστικού κλειδιού “movieId”, ώστε κάθε βαθμολογία να συνοδεύεται από τον τίτλο της ταινίας. Έτσι εξασφαλίζεται ότι μόνο ταινίες που υπάρχουν στο monies θα παραμείνουν.

Στη συνέχεια προχωράμε στην εξαγωγή βασικών στατιστικών χαρακτηριστικών του ενοποιημένου dataset, με στόχο την κατανόηση της δομής και της κατανομής των δεδομένων. Σε πρώτη φάση, εκτυπώνουμε ένα μικρό δείγμα των πρώτων πέντε γραμμών, επιβεβαιώνοντας έτσι ότι η συνένωση πραγματοποιήθηκε σωστά, και παράλληλα εκτελώντας τον πρώτο ποιοτικό-οπτικό έλεγχο πριν προχωρήσουμε σε περαιτέρω αναλύσεις. Προχωρώντας, εξάγουμε κάποια συνοπτικά μεγέθη, όπως τον μέσο όρο των αξιολογήσεων και τους αριθμούς μοναδικών χρηστών και μοναδικών ταινιών, σε μία προσπάθεια να προβληθεί το εύρος του συστήματος και το μέγεθος του πίνακα χρηστών και αντικειμένων που θα χρησιμοποιήσουμε. Έπειτα, υπολογίζουμε σε πόσες αξιολογήσεις έχει προβεί ο κάθε χρήστης και πόσες αξιολογήσεις έχει λάβει κάθε ταινία, προσπαθώντας να προβάσουμε την μεταβλητότητα στην συμπεριφορά των χρηστών και την ασυμμετρία στην δημοτικότητα μεταξύ των ταινιών.

```
34 # Άθροιση ελλείψεων και διπλότυπων
35 print("\nΕλλειψείς τιμές ανα στήλη : ")
36 print(data.isna().sum())
37
38 print("\nΔιπλότυπες τιμές : ", data.duplicated().sum())
39
40 # Ποσοστό κενών κελιών στον πίνακα χρηστών-ταινιών
41 num_unique_users = data['userId'].nunique() #μοναδικοί χρήστες
42 num_unique_items = data['movieId'].nunique() #μοναδικές ταινίες
43 num_interactions = len(data) #πόσα ratings έχει συνολικά
44 sparsity = 1 - (num_interactions / (num_unique_users * num_unique_items)) #ποσοστό του πίνακα χωρίς βαθμολογίες
45 print("\nMatrix sparsity:", round(sparsity, 6))
```

Εικόνα 5: Έλεγχος ελλείψεων, διπλοτύπων και αραιότητας δεδομένων

Επιπλέον, βλέπουμε και έναν έλεγχο της ακεραιότητας και της ποιότητας των δεδομένων με τον εντοπισμό ελλειπών ή διπλότυπων εγγραφών, επισημαίνοντας κάποια πιθανή ανάγκη για πρόσθετο καθαρισμό.

Από τα σημαντικότερα σημεία της ανάλυσης αποτελεί και ο προσδιορισμός της αραιότητας του ενοποιημένου πίνακα χρηστών-ταινιών. Ο υπολογισμός της προκύπτει μέσα από τον λόγο των υπαρκτών ratings προς το γινόμενο μοναδικών χρηστών και ταινιών και

εκτιμά το ποσοστό των πιθανών ζευγών χρήστη-ταινίας που δεν έχουν αξιολόγηση. Σε ένα σύστημα προτάσεων είναι αδύνατον κάθε χρήστης να έχει δει κάθε ταινία, για αυτό και ο πίνακας είναι αναπόφευκτα γεμάτος κενά.

```

48 # Οπτικοποίηση
49
50 # Γράφημα για ratings (συχνότητα εμφάνισης κάθε βαθμολογίας 1-5)
51 plt.figure(figsize=(6,4))
52 plt.hist(data['rating'], bins=10, edgecolor='black')
53 plt.title("Rating Distribution")
54 plt.xlabel("Rating")
55 plt.ylabel("Count")
56 plt.show()
57
58 # Οι 20 πιο πολυαξιολογημένες ταινίες
59 top_movies = data['title'].value_counts().head(20)
60 plt.figure(figsize=(12,5))
61 top_movies.plot(kind='bar')
62 plt.title("Top 20 Most Rated Movies")
63 plt.ylabel("Number of Ratings")
64 plt.show()

```

Εικόνα 6: Κατανομή βαθμολογιών και οι 20 πιο πολυαξιολογημένες ταινίες

Η ανάλυση ολοκληρώνεται με το κομμάτι της οπτικοποίησης μέσα από την παραγωγή γραφικών απεικονίσεων, οι οποίες λειτουργούν συμπληρωματικά προς την άμεση κατανόηση των μοτίβων και των ασυνεπειών των δεδομένων. Αρχικά, δημιουργείται ένα ιστόγραμμα της στήλης rating, με στόχο να αποτυπωθεί η κατανομή των βαθμολογιών στο σύνολο του dataset. Η εντολή `plt.figure(figsize=(6,4))` ορίζει το μέγεθος του γραφήματος, ενώ η `plt.hist(data['rating'], bins=10, edgecolor='black')` κατασκευάζει το ιστόγραμμα, χωρίζοντας το εύρος των τιμών της βαθμολογίας σε 10 διαστήματα και εμφανίζοντας τη συχνότητα εμφάνισης κάθε τιμής. Η προσθήκη περιγράμματος στις μπάρες μέσω του `edgecolor='black'` βελτιώνει την αναγνωσιμότητα του γραφήματος. Στη συνέχεια, με τις εντολές `plt.title("Rating Distribution")`, `plt.xlabel("Rating")` και `plt.ylabel("Count")` προστίθενται ο τίτλος και οι ετικέτες των αξόνων, ώστε το γράφημα να είναι πλήρως ερμηνεύσιμο. Η τελική εντολή `plt.show()` εμφανίζει το αποτέλεσμα. Η συγκεκριμένη απεικόνιση είναι ιδιαίτερα χρήσιμη,

καθώς επιτρέπει να φανεί αν οι χρήστες τείνουν να δίνουν κυρίως υψηλές, μέσες ή χαμηλές βαθμολογίες, άρα να εντοπιστεί η γενική συμπεριφορά αξιολόγησης στο dataset.

Προχωρώντας παρακάτω, δημιουργείται ένα δεύτερο γράφημα που παρουσιάζει τις 20 ταινίες με τον μεγαλύτερο αριθμό αξιολογήσεων. Σε πρώτο πλάνο, με την εντολή `top_movies = data['title'].value_counts().head(20)` υπολογίζεται πόσες φορές εμφανίζεται κάθε τίτλος ταινίας στο dataset και επιλέγονται οι 20 πρώτοι τίτλοι με τη μεγαλύτερη συχνότητα. Έπειτα, με την `plt.figure(figsize=(12,5))` ορίζεται ένα μεγαλύτερο μέγεθος σχήματος, κατάλληλο για την απεικόνιση πολλών κατηγοριών στον οριζόντιο άξονα, και με την `top_movies.plot(kind='bar')` παράγεται ραβδόγραμμα που εμφανίζει τις συγκεκριμένες ταινίες και το πλήθος των αξιολογήσεών τους. Ο τίτλος του γραφήματος προστίθεται με την `plt.title("Top 20 Most Rated Movies")`, ενώ η `plt.ylabel("Number of Ratings")` ορίζει την ετικέτα του κατακόρυφου άξονα. Τέλος, η `plt.show()` εμφανίζει το γράφημα. Η συγκεκριμένη οπτικοποίηση είναι σημαντική, διότι αναδεικνύει ποιες ταινίες συγκεντρώνουν τον μεγαλύτερο όγκο αλληλεπιδράσεων από τους χρήστες και επιτρέπει να εντοπιστούν οι πιο δημοφιλείς ή πιο συχνά αξιολογημένες επιλογές του dataset. Με τον τρόπο αυτό συμπληρώνεται η περιγραφική ανάλυση των δεδομένων, καθώς δεν εξετάζεται μόνο η συνολική κατανομή των βαθμολογιών, αλλά και η συγκέντρωση της δραστηριότητας γύρω από συγκεκριμένα αντικείμενα.

```
66 # Διανομή αριθμού βαθμολογιών ανά χρήστη σε δείγμα
67 sample_users = ratings_per_user.sample(min(10000, len(ratings_per_user)), random_state=42)
68 plt.figure(figsize=(6,4))
69 plt.hist(sample_users, bins=30, edgecolor='black')
70 plt.title("Number of Ratings per User")
71 plt.xlabel("Number of Ratings")
72 plt.ylabel("Count")
73 plt.show()
74
75 # Διανομή αριθμού βαθμολογιών ανά ταινία σε δείγμα
76 sample_movies = ratings_per_movie.sample(min(10000, len(ratings_per_movie)), random_state=42)
77 plt.figure(figsize=(6,4))
78 plt.hist(sample_movies, bins=30, edgecolor='black')
79 plt.title("Number of Ratings per Movie")
80 plt.xlabel("Number of Ratings")
81 plt.ylabel("Count")
82 plt.show()
```

Εικόνα 7: Κατανομή βαθμολογιών ανά χρήστη και ανά ταινία

Η ανάλυση συνεχίζεται με δύο ακόμη οπτικοποιήσεις, οι οποίες εξετάζουν τη διανομή του αριθμού βαθμολογιών ανά χρήστη και ανά ταινία. Στην πρώτη περίπτωση, ο κώδικας

δημιουργεί αρχικά ένα τυχαίο δείγμα από τη μεταβλητή `ratings_per_user` με την εντολή `sample(min(10000, len(ratings_per_user)), random_state=42)`. Αυτό σημαίνει ότι επιλέγονται έως 10.000 χρήστες από το σύνολο, ή λιγότεροι αν οι διαθέσιμοι χρήστες είναι λιγότεροι από αυτό το πλήθος. Η χρήση δειγματοληψίας γίνεται για λόγους καλύτερης διαχείρισης και πιο καθαρής οπτικοποίησης, ενώ η παράμετρος `random_state=42` εξασφαλίζει ότι το ίδιο δείγμα μπορεί να αναπαραχθεί σε επόμενες εκτελέσεις. Στη συνέχεια, με την `plt.figure(figsize=(6,4))` ορίζεται το μέγεθος του γραφήματος και με την `plt.hist(sample_users, bins=30, edgecolor='black')` παράγεται ιστόγραμμα που δείχνει πώς κατανέμεται ο αριθμός βαθμολογιών ανά χρήστη στο επιλεγμένο δείγμα. Οι 30 κάδοι (`bins=30`) επιτρέπουν μια πιο λεπτομερή απεικόνιση της κατανομής, ενώ ο τίτλος και οι ετικέτες των αξόνων διευκρινίζουν ότι στον οριζόντιο άξονα αποτυπώνεται ο αριθμός βαθμολογιών που έχει δώσει κάθε χρήστης και στον κατακόρυφο το πλήθος των χρηστών που εμφανίζουν αυτή τη συμπεριφορά. Η συγκεκριμένη οπτικοποίηση είναι χρήσιμη, επειδή δείχνει αν το dataset χαρακτηρίζεται από σχετικά ισορροπημένη δραστηριότητα ή αν, αντίθετα, ένα μικρό μέρος των χρηστών είναι υπερβολικά ενεργό ενώ οι περισσότεροι έχουν λίγες μόνο αξιολογήσεις.

Αντίστοιχα, στη δεύτερη περίπτωση εξετάζεται η διανομή του αριθμού βαθμολογιών ανά ταινία. Ο κώδικας δημιουργεί πρώτα ένα τυχαίο δείγμα από τη μεταβλητή `ratings_per_movie`, πάλι μέχρι 10.000 εγγραφές και με σταθερό `random_state=42`, και στη συνέχεια παράγει νέο ιστόγραμμα με την `plt.hist(sample_movies, bins=30, edgecolor='black')`. Το γράφημα αυτό δείχνει πώς κατανέμεται ο αριθμός αξιολογήσεων που έχει λάβει κάθε ταινία στο δείγμα. Με άλλα λόγια, επιτρέπει να φανεί αν οι βαθμολογίες κατανέμονται σχετικά ομοιόμορφα στο σύνολο των ταινιών ή αν η δραστηριότητα συγκεντρώνεται σε μικρό αριθμό ιδιαίτερα δημοφιλών τίτλων. Η οπτικοποίηση αυτή είναι ιδιαίτερα σημαντική σε προβλήματα recommendation, καθώς βοηθά να διαπιστωθεί ο βαθμός ανισοκατανομής του dataset: συνήθως λίγες ταινίες συγκεντρώνουν πολύ μεγάλο αριθμό ratings, ενώ πολλές άλλες εμφανίζονται πολύ πιο αραιά. Συνολικά, τα δύο αυτά γραφήματα λειτουργούν συμπληρωματικά, καθώς το πρώτο δείχνει την ένταση της δραστηριότητας από την πλευρά των χρηστών και το δεύτερο τη συγκέντρωση των αλληλεπιδράσεων από την πλευρά των ταινιών.

```

85 # Δείγμα 20000 για αποφυγή υπερφόρτωσης της μνήμης
86 # Δειγματοληψία δεδομένων
87 sample_data = data.sample(n=20000, random_state=42)
88
89 # Pivot πίνακας
90 sample_matrix = sample_data.pivot(index='userId', columns='movieId', values='rating')
91
92 # Μετατροπή σε sparse format
93 sparse_matrix = sample_matrix.notna()
94
95 plt.figure(figsize=(12, 6))
96
97 # spy plot - δείχνει μόνο πού υπάρχει rating
98 plt.spy(sparse_matrix, markersize=2, aspect='auto')
99
100 plt.title("Sparse User-Movie Matrix (Ratings Positions)")
101 plt.xlabel("Movie Index")
102 plt.ylabel("User Index")
103
104 plt.show()

```

Εικόνα 8: Heatmap πίνακα χρηστών-ταινιών

Τέλος, πραγματοποιείται μια οπτική αποτύπωση της αραιότητας του πίνακα χρηστών-ταινιών, ώστε να γίνει πιο άμεσα κατανοητή η δομή του dataset ως προς τις διαθέσιμες αλληλεπιδράσεις. Αρχικά, ο κώδικας δημιουργεί ένα τυχαίο δείγμα από το πλήρες σύνολο δεδομένων με την εντολή `sample_data = data.sample(20000, random_state=42)`. Η δειγματοληψία αυτή γίνεται για να αποφευχθεί υπερβολική κατανάλωση μνήμης και να καταστεί εφικτή η οπτικοποίηση, καθώς το πλήρες dataset είναι πολύ μεγάλο. Στη συνέχεια, με την εντολή `sample_data.pivot(index='userId', columns='movieId', values='rating')` δημιουργείται ένας πίνακας όπου κάθε γραμμή αντιστοιχεί σε έναν χρήστη, κάθε στήλη σε μία ταινία και κάθε κελί περιέχει τη βαθμολογία, εφόσον υπάρχει. Με αυτόν τον τρόπο σχηματίζεται μια τυπική user-movie matrix στο επίπεδο του δείγματος.

Στη συνέχεια, με την εντολή `sample_matrix.notna()` ο πίνακας μετατρέπεται σε δυαδική μορφή, όπου δεν ενδιαφέρει πλέον η τιμή της βαθμολογίας αλλά μόνο το αν υπάρχει ή όχι αξιολόγηση. Δηλαδή, κάθε θέση του πίνακα παίρνει ουσιαστικά τη μορφή True/False, ανάλογα με το αν έχει καταγραφεί κάποια αλληλεπίδραση χρήστη-ταινίας. Στη συνέχεια, με την `plt.spy(sparse_matrix, markersize=2, aspect='auto')` παράγεται μια οπτική αναπαράσταση των θέσεων στις οποίες υπάρχουν ratings. Ο τίτλος και οι ετικέτες των αξόνων διευκρινίζουν ότι

στον οριζόντιο άξονα αποτυπώνονται οι θέσεις των ταινιών και στον κατακόρυφο οι θέσεις των χρηστών μέσα στον πίνακα.

Η συγκεκριμένη οπτικοποίηση είναι ιδιαίτερα σημαντική, επειδή επιτρέπει να φανεί με άμεσο τρόπο η αραιότητα του πίνακα χρηστών-ταινιών. Σε datasets συστημάτων προτάσεων, ο πίνακας αυτός είναι συνήθως εξαιρετικά sparse, δηλαδή το μεγαλύτερο μέρος των πιθανών συνδυασμών χρηστών και ταινιών δεν περιέχει αξιολόγηση. Το spy plot βοηθά να επιβεβαιωθεί οπτικά αυτή η ιδιότητα, δείχνοντας ότι οι καταγεγραμμένες βαθμολογίες εμφανίζονται μόνο σε ένα πολύ μικρό ποσοστό των συνολικών θέσεων του πίνακα. Η πληροφορία αυτή είναι σημαντική, επειδή η υψηλή αραιότητα αποτελεί βασικό χαρακτηριστικό των προβλημάτων recommendation και επηρεάζει άμεσα τόσο την επιλογή της μεθόδου όσο και τη δυσκολία της πρόβλεψης προτιμήσεων.

3.1.2 Παρουσίαση αποτελεσμάτων αρχείου eda.py

```
Φόρτωση βαθμολογιών: (32000204, 4)
Φόρτωση ταινιών: (87585, 3)

Συνένωση δεδομένων: (32000204, 5)

Δείγμα πρώτων γραμμών:
   userId  ... title
0      1  ... Sense and Sensibility (1995)
1      1  ... Leaving Las Vegas (1995)
2      1  ... City of Lost Children, The (Cité des enfants p...
3      1  ... Shanghai Triad (Yao a yao yao dao waipo qiao) ...
4      1  ... Twelve Monkeys (a.k.a. 12 Monkeys) (1995)

[5 rows x 5 columns]

Μέση Βαθμολογία: 3.54
Αριθμός Μοναδικών Χρηστών: 200948
Αριθμός Μοναδικών Ταινιών: 84432

Αξιολογήσεις ανα χρήστη:
count    200948.000000
mean       159.246193
std       282.025462
min        20.000000
25%        36.000000
50%        73.000000
75%       167.000000
max      33332.000000
Name: rating, dtype: float64
```

Εικόνα 9: Βασικά στατιστικά της βάσης δεδομένων

Όπως μπορούμε να δούμε, τα δεδομένα μας περιλαμβάνουν 32.000.204 αξιολογήσεις και 87.585 ταινίες, γεγονός που καταδεικνύει ότι το dataset είναι μεγάλης κλίμακας. Κατά την

συνένωση δεν παρατηρήθηκε μείωση στον αριθμό γραμμών πράγμα που δείχνει δομική συνέπεια μεταξύ των δεδομένων του dataset.

Έπειτα, ακολουθεί η μέση βαθμολογία με τιμή 3,54. Από την τιμή αυτή προκύπτει μια θετική μεροληψία, καθώς γίνεται αντιληπτό ότι οι χρήστες τείνουν να βαθμολογούν πιο συχνά ταινίες που τους άρεσαν αποφεύγοντας το περιεχόμενο που εξ' αρχής θεωρούσαν αδιάφορο. Επιπλέον, ο αριθμός μοναδικών ταινιών είναι 84.432 και μοναδικών χρηστών 200.948. Το μεγάλο πλήθος ταινιών και χρηστών οδηγεί στην εκθετική αύξηση του χώρου πιθανών αλληλεπιδράσεων κάτι που εξηγεί τη υψηλή αραιότητα που βλέπουμε και παρακάτω.

Συνεχίζοντας την ανάλυση, προχωράμε στις αξιολογήσεις ανά χρήστη. Ο μέσος χρήστης έχει αξιολογήσει 159 ταινίες με τυπική απόκλιση 282, γεγονός που δείχνει την μεγάλη διασπορά, καθώς υπάρχουν χρήστες που έχουν βαθμολογήσει πολύ περισσότερες ή πολύ λιγότερες ταινίες από τον μέσο όρο. Έπειτα βλέπουμε τον χρήστη με τις λιγότερες αξιολογήσεις να έχει βαθμολογήσει 20 ταινίες και με τις περισσότερες να έχει βαθμολογήσει 33.332 ταινίες, υποδεικνύοντας έτσι την ύπαρξη ακραίων τιμών, γεγονός που μπορεί να επηρεάσει την ακρίβεια των μοντέλων που θέλουμε να υλοποιήσουμε. Αναλύοντας και τα τεταρτημόρια που προκύπτουν αντιλαμβανόμαστε ότι η πλειονότητα των χρηστών έχουν λίγες, σχετικά, αξιολογήσεις ενώ λίγοι χρήστες παρουσιάζουν μεγάλο αριθμό αξιολογήσεων.

```
Ratings per movie :
count      84432.000000
mean       379.005638
std        2592.439791
min         1.000000
25%         2.000000
50%         5.000000
75%        25.000000
max       102929.000000
Name: rating, dtype: float64

Ελλιπείς τιμές ανα στήλη :
userId      0
movieId     0
rating      0
timestamp   0
title       0
dtype: int64

Διπλότυπες τιμές : 0

Matrix sparsity: 0.998114

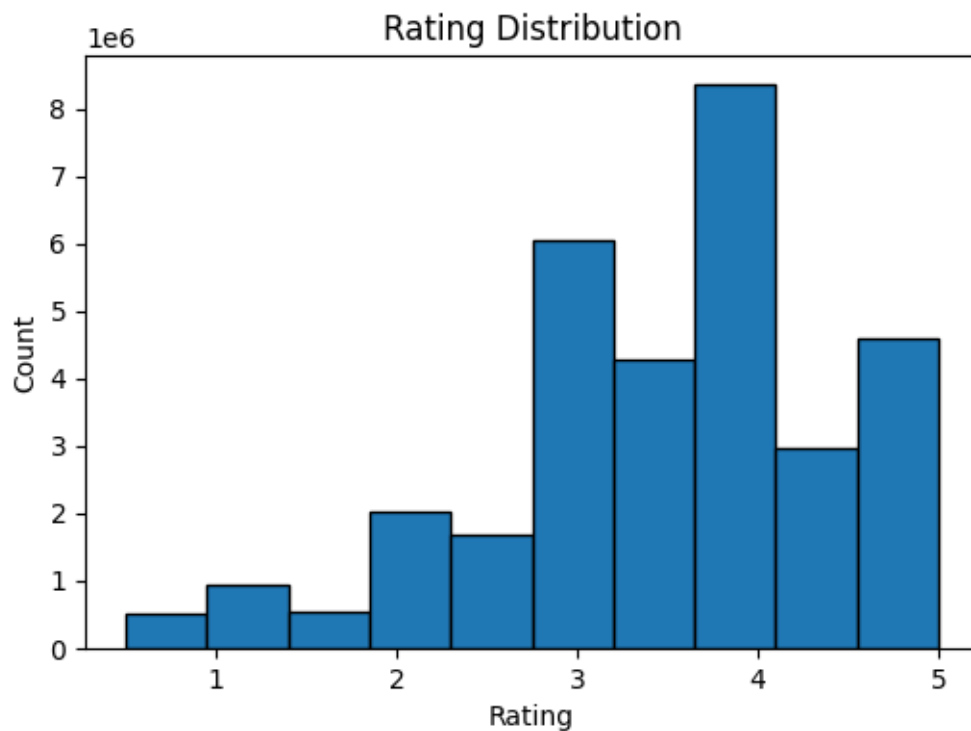
Process finished with exit code 0
```

Εικόνα 10: Έλεγχος ελλείψεων, διπλοτύπων και αραιότητας

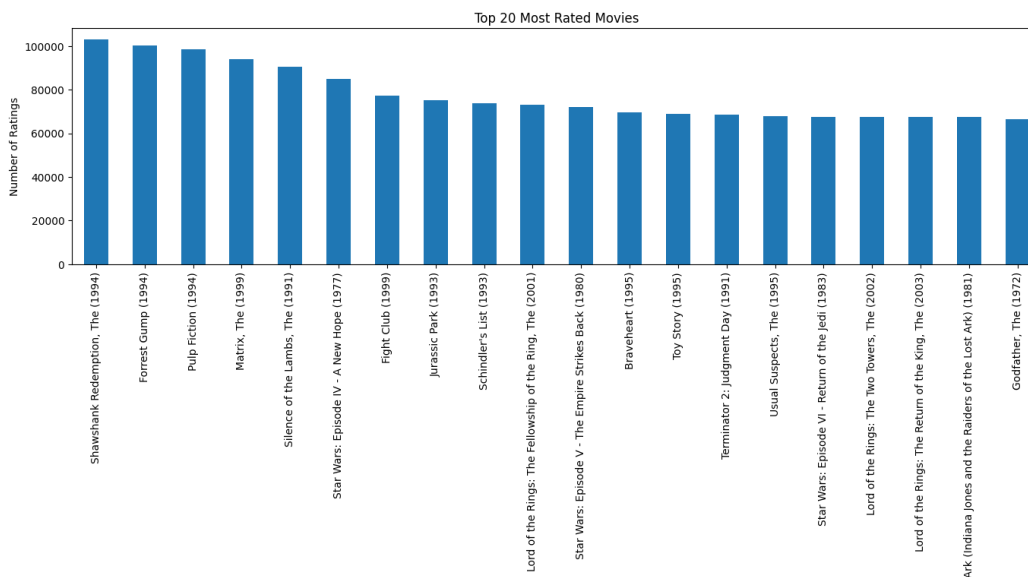
Περνώντας στις αξιολογήσεις ανά ταινία, κάθε ταινία έχει αξιολογηθεί 379 φορές κατά μέσο όρο και η τυπική απόκλιση είναι τεράστια με τιμή 2.592, γεγονός που υποδηλώνει ότι οι περισσότερες ταινίες έχουν πολύ λίγες αξιολογήσεις και υπάρχουν και μερικές που έχουν πάρα πολλές. Αναφορικά με τις περισσότερες και τις λιγότερες αξιολογήσεις ανά ταινία υπάρχει τουλάχιστον μια ταινία με 1 αξιολόγηση και τουλάχιστον μία ταινία με 102.929 αξιολογήσεις. Τέλος, από την πλευρά των τεταρτημόριων, παρατηρούμε ότι η διάμεσος είναι μόλις 5. Αυτό σημαίνει ότι οι μισές ταινίες έχουν από 1 έως 5 αξιολογήσεις. Όπως εύκολα αντιλαμβανόμαστε, οι περισσότεροι χρήστες δεν έχουν αξιολογήσει την πλειονότητα των ταινιών.

Συνοψίζοντας, η αναφορά για μηδενικές ελλιπείς και διπλότυπες τιμές τεκμηριώνει ότι δεν υπάρχει ανάγκη για επιπλέον καθαρισμό και επιβεβαιώνει την αξιοπιστία των δεδομένων μας.

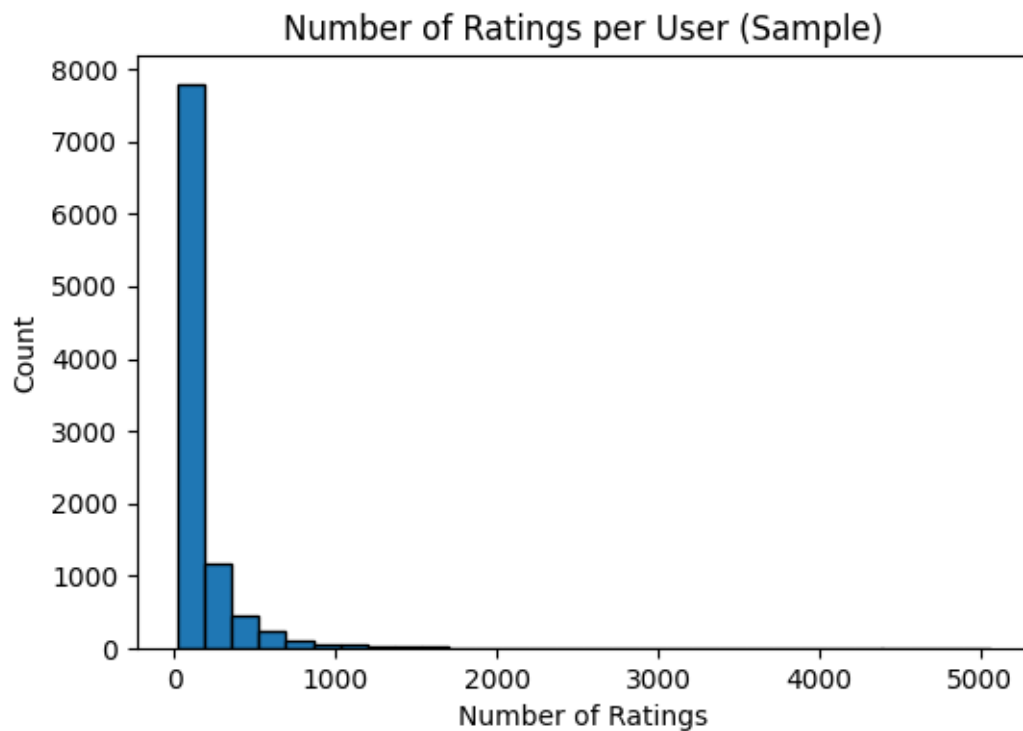
Παρακάτω ακολουθούν και κάποια ενδεικτικά σχεδιαγράμματα που παρήγαγε το σύστημα μας σχετικά με την βάση δεδομένων:



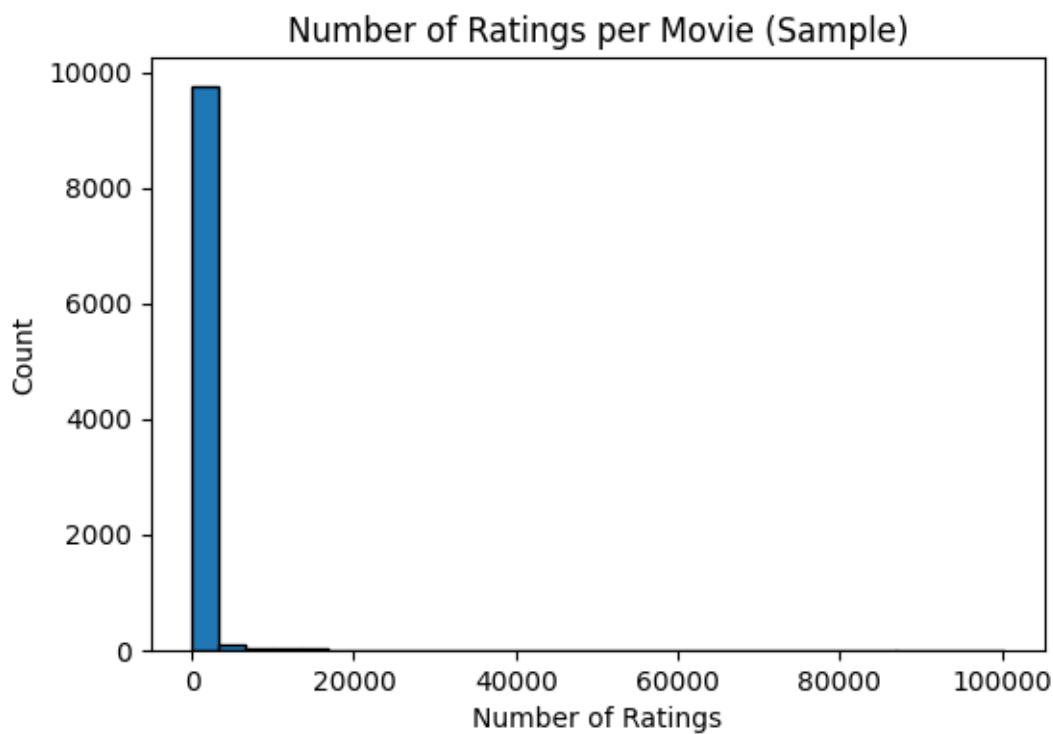
Εικόνα 11: Διάγραμμα Κατανομής Αξιολογήσεων



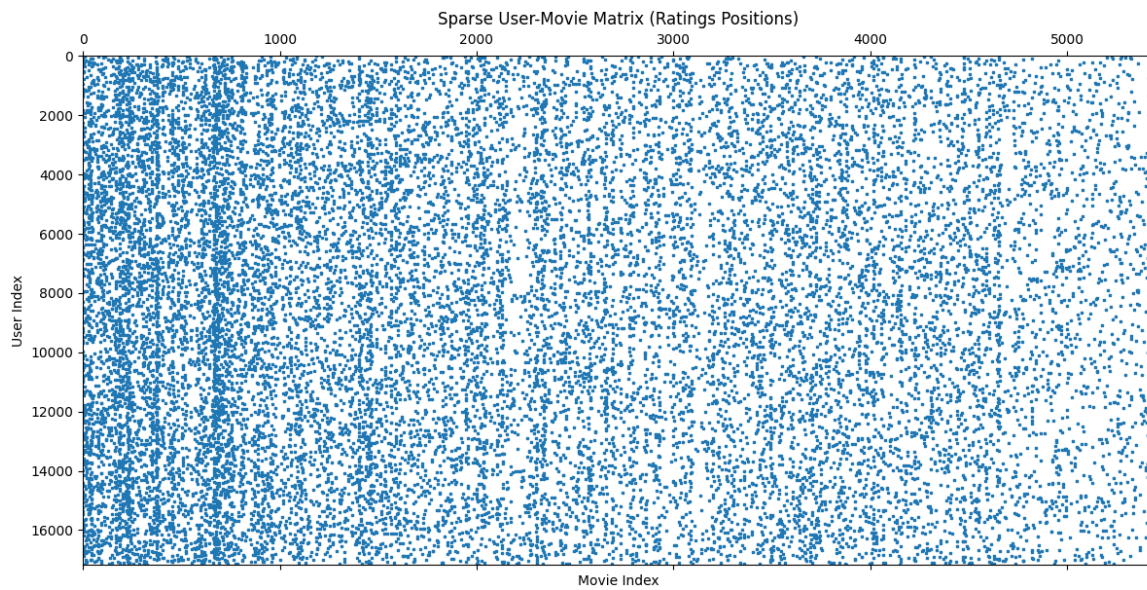
Εικόνα 12: Διάγραμμα των 20 πιο Πολυαξιολογημένων Ταινιών



Εικόνα 13: Διάγραμμα Αριθμού Αξιολογήσεων ανά Χρήστη

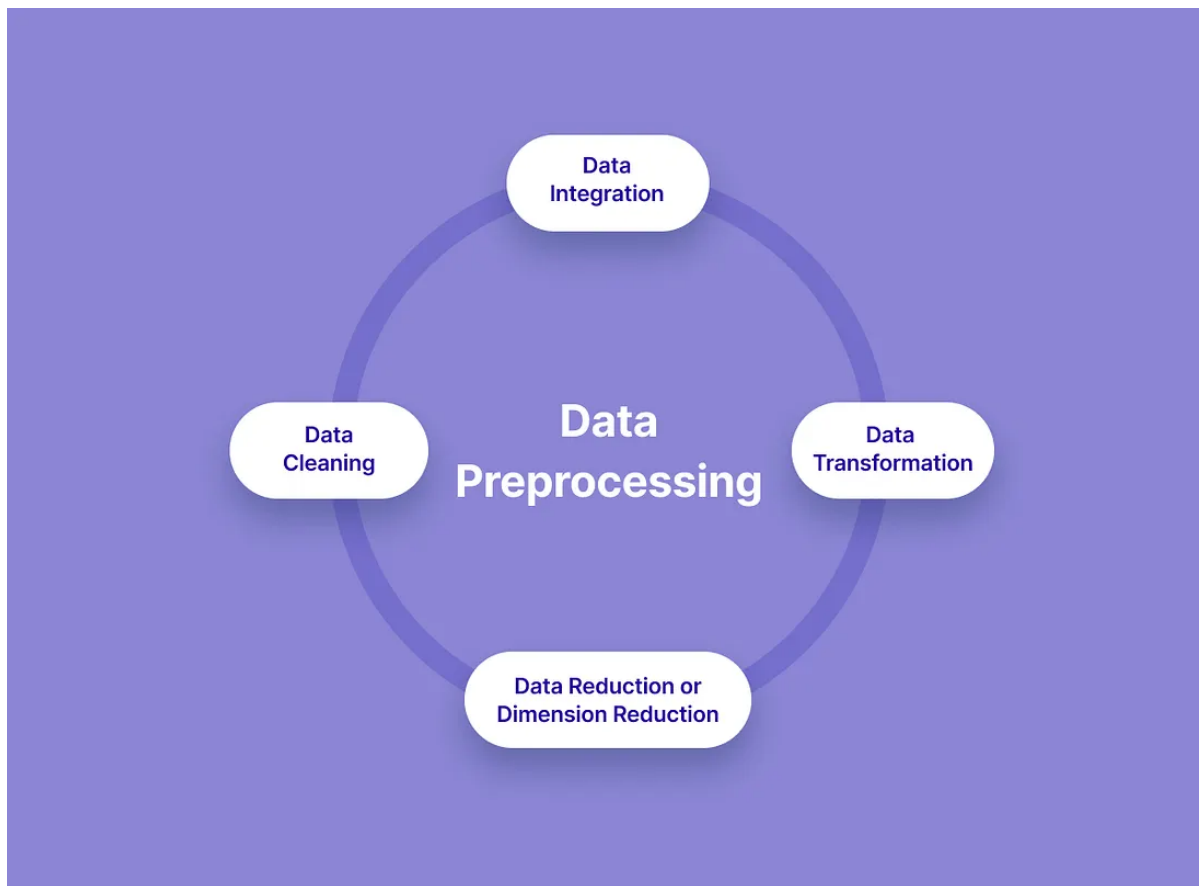


Εικόνα 14: Διάγραμμα Αριθμού Αξιολογήσεων ανά Ταινία



Εικόνα 15: Διάγραμμα Αραιότητας Αλληλεπιδράσεων Χρηστών - Ταινιών

3.2 Προεπεξεργασία και ανάλυση δεδομένων



Εικόνα 16: Διαδικασίες Προεπεξεργασίας, Πηγή: medium.com

Ακόμη και όταν συνδυάζουμε σωστά δεδομένα από διάφορες πηγές σε ένα dataset, δεν μπορούμε βεβαιωμένα να θεωρήσουμε ότι είναι τέλεια. Μπορεί να περιέχουν κενά, λανθασμένες τιμές ή ασυνεπή μορφοποίηση που επηρεάζει την ποιότητα των αποτελεσμάτων. Όσο περισσότερα προβληματικά δεδομένα περιέχει ένα σύνολο εκπαίδευσης (train set) ή δοκιμής (test set), τόσο λιγότερο αξιόπιστο θα είναι το μοντέλο που προκύπτει από αυτά. Στην πράξη, τα δεδομένα που χρειάζονται προσοχή είναι εμφανίζονται κυρίως με δύο μορφές. Ελλείποντα στοιχεία, τα οποία μπορούν να συμπληρωθούν ή ακόμη και να αγνοηθούν, και θορυβώδη στοιχεία τα οποία εντοπίζονται κατά την Διερευνητική ανάλυση δεδομένων. Στην περίπτωση μας, δεν έχουμε ελλείποντα στοιχεία ούτε διπλότυπα (García et al., 2015).

Ένα πολύ σημαντικό θέμα που πρέπει να λάβουμε υπόψη κατά το κομμάτι της προεπεξεργασίας είναι το πρόβλημα της ψυχρής εκκίνησης (cold-start problem). Η απουσία

αξιολογήσεων από έναν χρήστη ή η απουσία αξιολογήσεων σε μία ταινία αποτελούν προβλήματα τα οποία μπορούν να καταστήσουν αδύνατη την παραγωγή αξιόπιστων προτάσεων από ένα μοντέλο. Το πρόβλημα του αντικειμένου εμφανίζεται όταν ένα στοιχείο/ταινία το οποίο δεν έχει αξιολογηθεί αρκετά από τους χρήστες με αποτέλεσμα να μην παρέχει αρκετά δεδομένα στο μοντέλο και να προκαλεί υψηλή αβεβαιότητα στην πρόβλεψη του. Κάτι παρόμοιο ισχύει και με το πρόβλημα του χρήστη. Χρήστες με λίγες αξιολογήσεις δεν δίνουν αρκετή πληροφορία στα μοντέλα για να καθοριστεί η προτίμηση τους.

Άξιο αναφοράς είναι και το πρόβλημα της μακράς ουράς (long tail problem). Στα συστήματα προτάσεων, οι περισσότερες αξιολογήσεις συγκεντρώνονται σε λίγα, δημοφιλή αντικείμενα, ενώ πολλά άλλα έχουν πολύ λίγες αξιολογήσεις. Τα αντικείμενα με λίγες αξιολογήσεις είναι δύσκολο να προταθούν με αξιόπιστο τρόπο. Αφαιρώντας τις ουσιαστικά ανύπαρκτες ταινίες, τα μοντέλα μαθαίνουν από ταινίες που έχουν αρκετά δεδομένα, μειώνοντας έτσι αρκετά τον θόρυβο και βελτιώνοντας την αξιοπιστία των προτάσεων (Bobadilla et al., 2012).

3.2.1 Ανάλυση αρχείου preprocess.py

```
1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3
4 # Load Ratings and Movies
5 ratings_path = '../data/ml-32m/ratings.csv'
6 movies_path = '../data/ml-32m/movies.csv'
7
8 # load ratings
9 ratings_cols = ['userId', 'movieId', 'rating', 'timestamp']
10 ratings = pd.read_csv(ratings_path, usecols=ratings_cols)
11
12 # convert timestamp to datetime
13 ratings['timestamp'] = pd.to_datetime(ratings['timestamp'], unit='s')
14 print("Ratings loaded:", ratings.shape)
15
16 # load movies
17 movies = pd.read_csv(movies_path)
18
19 # split genres into list
20 if 'genres' in movies.columns:
21     movies['genres'] = movies['genres'].str.split('|')
22
23 print("Movies loaded:", movies.shape)
24
25 # Merge Ratings with Movie Info
26 data = ratings.merge(movies[['movieId', 'title', 'genres']], on='movieId', how='left')
27 print("Ratings merged with movies:", data.shape)
```

Εικόνα 17: Φόρτωση και συνένωση των δεδομένων

Ο παραπάνω κώδικας υλοποιεί το στάδιο της προεπεξεργασίας και οργάνωσης των δεδομένων μας, γεγονός το οποίο αποτελεί κρίσιμο σημείο για την ανάπτυξη αλλά και αξιολόγηση συστημάτων προτάσεων. Η διαδικασία βασίζεται στην βάση δεδομένων μας και περιλαμβάνει δράσεις όπως η φόρτωση, ο καθαρισμός, το φιλτράρισμα, ο εμπλουτισμός και ο διαχωρισμός των δεδομένων μας σε δύο σύνολα εκπαίδευσης (train set) και ελέγχου (test set).

Ξεκινώντας, φορτώνονται τα δεδομένα αξιολογήσεων από το αρχείο ratings.csv, φορτώνοντας τις στήλες που διαθέτει το συγκεκριμένο αρχείο. Η μετατροπή της χρονικής σήμανσης από unix timestamp σε αντικείμενο ημερομηνίας (datetime) διευκολύνει πιθανές μελλοντικές χρονικές αναλύσεις, ενώ δεν επηρεάζει την βασική λειτουργία του παρόντος συστήματος. Στην συνέχεια φορτώνονται και τα δεδομένα ταινιών από το αρχείο movies.csv, το οποίο περιλαμβάνει τίτλους και είδη ταινιών.

Τα είδη ταινιών, τα οποία αρχικά είναι αποθηκευμένα ως συμβολοσειρά (string) με διαχωριστικό τον χαρακτήρα «|», μετατρέπονται σε λίστα. Η συγκεκριμένη μετατροπή είναι

ιδιαίτερα χρήσιμη καθώς μετατρέπει την ευκολότερη επεξεργασία και μετατροπή των genres σε διανύσματα σε επόμενα στάδια (π.χ. TF-IDF). Εν συνεχεία, τα δεδομένα αξιολογήσεων εμπλουτίζονται με τα δεδομένα ταινιών μέσω συγχώνευσης των δύο αρχείων βάσει του movieId, δημιουργώντας έτσι ένα ενιαίο σύνολο δεδομένων με συμπεριφορικά δεδομένα χρηστών και με πληροφορίες για το περιεχόμενο των ταινιών.

```
29 # Filter Sparse Users / Movies (fixed)
30 min_ratings_user = 20
31 min_ratings_movie = 10
32
33 # get users with enough ratings
34 users_to_keep = data['userId'].value_counts()[data['userId'].value_counts() >= min_ratings_user].index
35 # get movies with enough ratings
36 movies_to_keep = data['movieId'].value_counts()[data['movieId'].value_counts() >= min_ratings_movie].index
37
38 # filter and make a copy to avoid SettingWithCopyWarning
39 data_filtered = data[data['userId'].isin(users_to_keep) & data['movieId'].isin(movies_to_keep)].copy()
40 print("\nData after filtering sparse users/movies:", data_filtered.shape)
41
42 # Encode User and Movie IDs
43 user2idx = {user: idx for idx, user in enumerate(data_filtered['userId'].unique())}
44 movie2idx = {movie: idx for idx, movie in enumerate(data_filtered['movieId'].unique())}
45
46 data_filtered['user_idx'] = data_filtered['userId'].map(user2idx)
47 data_filtered['movie_idx'] = data_filtered['movieId'].map(movie2idx)
48 print("\nUser and Movie IDs encoded")
49
50 # Split into Train / Test
51 train, test = train_test_split(*arrays: data_filtered, test_size=0.2, random_state=42)
52 print("\nTrain shape:", train.shape)
53 print("\nTest shape:", test.shape)
54
55 # Save Preprocessed Data
56 train.to_csv('../data/ml-32m/train.csv', index=False)
57 test.to_csv('../data/ml-32m/test.csv', index=False)
58 print("\nPreprocessed data saved to CSV")
```

Εικόνα 18: Φιλτράρισμα, κωδικοποίηση και διαχωρισμός δεδομένων

Έπειτα ακολουθεί ένα σημαντικό στάδιο, αυτό του φιλτραρίσματος αραιών δεδομένων. Ορίζεται κατώφλι ελάχιστου αριθμού αξιολογήσεων τόσο για τους χρήστες όσο και για τις ταινίες. Πιο συγκεκριμένα, διατηρούνται μόνο οι χρήστες που έχουν καταχωρήσει τουλάχιστον 20 αξιολογήσεις και οι ταινίες που έχουν λάβει τουλάχιστον 10 αξιολογήσεις. Από την εξερευνητική ανάλυση δεδομένων προέκυψε ότι ο ελάχιστος αριθμός αξιολογήσεων ανά χρήστη στο αρχικό σύνολο δεδομένων είναι 20. Παρότι το φίλτρο των ελάχιστων αξιολογήσεων χρηστών δεν μεταβάλλει επί της ουσίας τα συγκεκριμένα δεδομένα, διατηρείται στον κώδικα για λόγους μεθοδολογικής πληρότητας και γενίκευσης της μεθοδολογίας της διαδικασίας. Αναφορικά τώρα με το φίλτρο ελάχιστων αξιολογήσεων ανά ταινία, αν και το 50% των ταινιών διαθέτει έως 5 αξιολογήσεις, οι συγκεκριμένες ταινίες αντιστοιχούν σε μικρό

ποσοστό του συνολικού αριθμού των αξιολογήσεων. Η εφαρμογή κατωφλιού ίσου με 10 απομακρύνει κυρίως ακραία αραιά αντικείμενα, μειώνοντας τον θόρυβο και βελτιώνοντας τη σταθερότητα της αξιολόγησης, χωρίς να επηρεάζει ουσιαστικά τον όγκο και την αντιπροσωπευτικότητα των δεδομένων. Συνεχίζοντας, το φιλτραρισμένο σύνολο δεδομένων αποθηκεύεται σε νέα μεταβλητή και συνεχίζουμε με την επεξεργασία.

Επιπλέον, πραγματοποιείται κωδικοποίηση των αναγνωριστικών χρηστών (userId) και ταινιών (movieId) σε αριθμητικούς δείκτες, κατάλληλους για χρήση σε αλγόριθμους μηχανικής μάθησης. Οι δείκτες αυτοί προστίθενται ως νέες στήλες στο σύνολο δεδομένων και αποτελούν την εσωτερική αναπαράσταση χρηστών και αντικειμένων στα επόμενα στάδια της ανάλυσης.

Το επόμενο σημείο αφορά τον διαχωρισμό των δεδομένων σε train set και test set, με αναλογία 80% - 20%. Ο διαχωρισμός πραγματοποιείται τυχαία αλλά με σταθερό seed ώστε όλες οι μέθοδοι να αξιολογούνται στα ίδια δεδομένα μεταξύ διαφορετικών εκτελέσεων, εξασφαλίζοντας έτσι την αναπαραγωγιμότητα του πειράματος. Το train set χρησιμοποιείται για την εκμάθηση των μοντέλων, ενώ το test set για την αξιολόγηση του μοντέλου. Τέλος, τα σύνολα δεδομένων αποθηκεύονται σε δύο ξεχωριστά αρχεία csv.

```
Ratings loaded: (32000204, 4)
Movies loaded: (87585, 3)
Ratings merged with movies: (32000204, 6)

Data after filtering sparse users/movies: (31842705, 6)

User and Movie IDs encoded

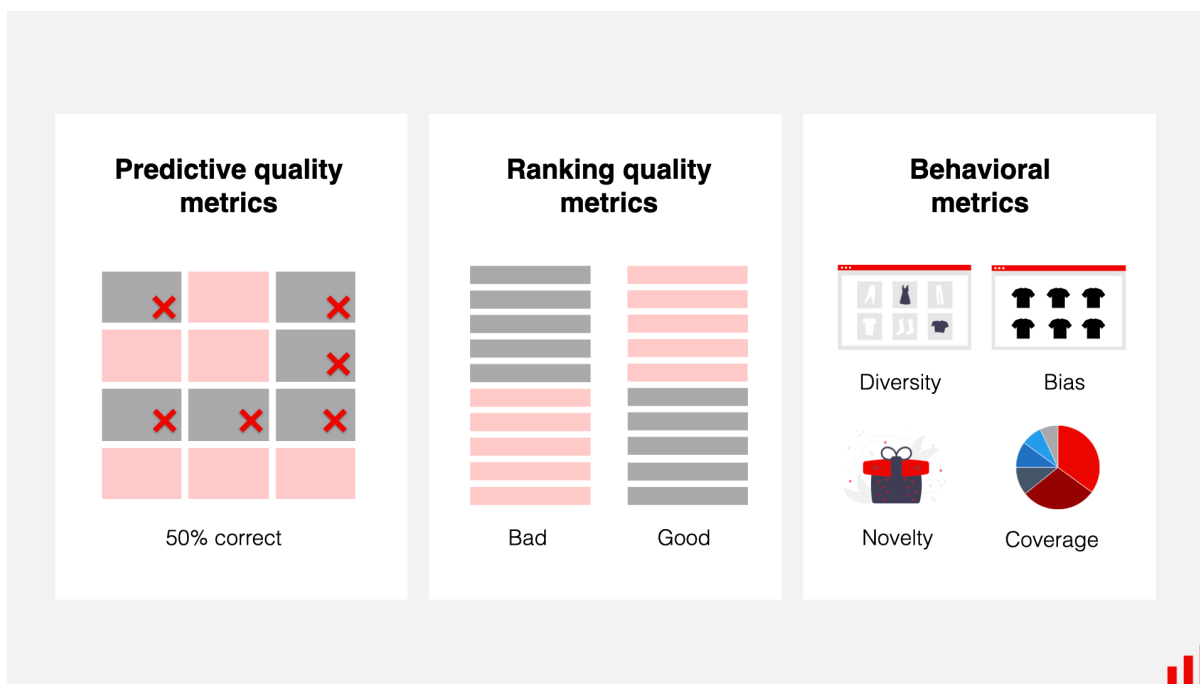
Train shape: (25474164, 8)
Test shape: (6368541, 8)

Preprocessed data saved to CSV

Process finished with exit code 0
```

Εικόνα 19: Αποτελέσματα Προεπεξεργασίας Δεδομένων

3.3 Μετρικές Αξιολόγησης



Εικόνα 20: Κατηγορίες Μετρικών Αξιολόγησης, Πηγή: evidentlyai.com

Η αξιολόγηση των συστημάτων σύστασης αποτελεί σύνθετη διαδικασία, καθώς η απόδοση ενός αλγορίθμου επηρεάζεται σημαντικά από τα χαρακτηριστικά του εκάστοτε dataset, όπως το πλήθος χρηστών και αντικειμένων, η πυκνότητα των αξιολογήσεων και η κλίμακα βαθμολόγησης. Ακόμη, η ποιότητα ενός recommender system δεν μπορεί να αποτυπωθεί πλήρως από μία μόνο μετρική, καθώς διαφορετικές μετρικές εξετάζουν διαφορετικές πτυχές της απόδοσης. Έτσι, στην παρούσα εργασία δεν χρησιμοποιείται μία μετρική πρόβλεψης rating, όπως το MAE ή το RMSE, αλλά ένα σύνολο ranking-oriented μετρικών, όπως Precision@10, Recall@10, MAP@10, NDCG@10 και HitRate@10, οι οποίες είναι πιο κατάλληλες για την αξιολόγηση λιστών top-k προτάσεων (Herlocker et al., 2004). Πιο συγκεκριμένα, οι μετρικές αυτές επιλέγονται επειδή το ζητούμενο της παρούσας εργασίας δεν είναι να προβλεφθεί με ακρίβεια η αριθμητική βαθμολογία που θα έδινε ένας χρήστης σε μια ταινία, αλλά να αξιολογηθεί η ποιότητα της λίστας προτάσεων που παράγει το σύστημα. Με άλλα λόγια, το ενδιαφέρον εστιάζει στο αν οι ταινίες που εμφανίζονται στις πρώτες θέσεις είναι πράγματι σχετικές με τις προτιμήσεις του χρήστη και στο αν η σειρά κατάταξής τους είναι αποτελεσματική. Για τον λόγο αυτό χρησιμοποιούνται μετρικές top-k, όπου στην παρούσα εργασία το K έχει οριστεί ίσο με 10. Ως relevant αντικείμενα θεωρούνται οι ταινίες του test set που έχουν αξιολογηθεί από τον χρήστη με βαθμολογία

μεγαλύτερη ή ίση από το προκαθορισμένο κατώφλι αξιολόγησης. Κατά αυτό τον τρόπο, οι μετρικές αυτές συγκρίνουν τις δέκα προτεινόμενες ταινίες του συστήματος με τις ταινίες που θεωρούνται πραγματικά σχετικές για τον χρήστη.

Η μετρική Precision@K δείχνει το ποσοστό των προτεινόμενων αντικειμένων στις πρώτες K θέσεις που είναι πράγματι relevant. Ο τύπος της είναι:

$$\text{Precision@k} = \frac{\text{RelevantItems} \cap \text{RecommendedItems@K}}{k}$$

όπου RelevantItems είναι το σύνολο των σχετικών αντικειμένων για τον χρήστη, ενώ RecommendedItems είναι το σύνολο των K πρώτων αντικειμένων που προτάθηκαν στον χρήστη. Στην παρούσα εργασία, το Precision@10 δείχνει πόσες από τις δέκα ταινίες που προτείνει το σύστημα είναι τελικά σχετικές. Η μετρική αυτή αποτυπώνει την «καθαρότητα» της λίστας προτάσεων, δηλαδή το κατά πόσο οι περιορισμένες θέσεις του top-10 χρησιμοποιούνται για αντικείμενα που έχουν πραγματική αξία για τον χρήστη.

Η μετρική Recall@10 εξετάζει διαφορετική πλευρά της απόδοσης. Δεν μετρά πόσες από τις προτάσεις είναι σωστές, αλλά πόσα από όλα τα relevant αντικείμενα του χρήστη κατάφερε να ανακτήσει το σύστημα μέσα στις πρώτες K θέσεις. Ο τύπος της είναι:

$$\text{Recall@k} = \frac{|\text{RelevantItems} \cap \text{RecommendedItems@K}|}{|\text{RelevantItems}|}$$

Συνεπώς, το Recall@10 δείχνει το ποσοστό των σχετικών ταινιών του χρήστη που εμφανίστηκαν μέσα στις δέκα πρώτες προτάσεις. Η μετρική αυτή είναι χρήσιμη επειδή ένα σύστημα μπορεί να έχει σχετικά καλό Precision , αλλά να καλύπτει μόνο μικρό μέρος των συνολικά σχετικών αντικειμένων. Επομένως, το Precision και το Recall εξετάζονται συμπληρωματικά: το πρώτο δείχνει πόσο στοχευμένη είναι η λίστα, ενώ το δεύτερο δείχνει πόσο καλά καλύπτονται οι πραγματικές προτιμήσεις του χρήστη.

Για την αξιολόγηση σε πολλούς χρήστες, οι τιμές των παραπάνω μετρικών υπολογίζονται αρχικά ανά χρήστη και στη συνέχεια λαμβάνεται ο μέσος όρος τους. Έτσι, το μέσο Precision@K και το μέσο Recall@K υπολογίζονται ως εξής:

$$\text{Average Precision@K} = \frac{1}{U} \sum_{u=1}^U \frac{|\text{Relevant Items}_u \cap \text{Recommended Items}_u@K|}{K}$$

$$\text{Average Recall@K} = \frac{1}{|U|} \sum_{u \in U} \frac{|\text{Relevant Items}_u \cap \text{Recommended Items}_u@K|}{|\text{Relevant Items}_u|}$$

όπου U είναι το σύνολο των χρηστών που συμμετέχουν στην αξιολόγηση. Με αυτόν τον τρόπο, οι τελικές τιμές δεν εκφράζουν την απόδοση του συστήματος για έναν μόνο χρήστη, αλλά τη μέση συμπεριφορά του μοντέλου στο σύνολο του δείγματος αξιολόγησης.

Η *Mean Average Precision* — MAP είναι μια μετρική που λαμβάνει υπόψη όχι μόνο αν τα σχετικά αντικείμενα εμφανίζονται στις προτάσεις, αλλά και τη θέση στην οποία εμφανίζονται. Επομένως, δίνει μεγαλύτερη βαρύτητα στα σχετικά αντικείμενα που βρίσκονται ψηλότερα στη λίστα κατάταξης. Για κάθε χρήστη, η *Average Precision* υπολογίζεται ως ο μέσος όρος των τιμών Precision@K στα σημεία της λίστας όπου εμφανίζεται ένα σχετικό αντικείμενο. Στη συνέχεια, η MAP προκύπτει ως ο μέσος όρος αυτών των τιμών για όλους τους χρήστες:

$$\text{MAP} = \frac{1}{U} \sum_{u=1}^U \left(\frac{1}{|\text{Relevant Items}_u|} \sum_{k=1}^{|\text{Recommended Items}_u|} \text{Precision@k} \times \text{rel}_u(k) \right)$$

όπου $\text{rel}_u(k)$ είναι μια δυαδική συνάρτηση ένδειξης, η οποία παίρνει τιμή 1 όταν το αντικείμενο στη θέση k είναι σχετικό για τον χρήστη u , και τιμή 0 όταν δεν είναι σχετικό. Με απλά λόγια, η MAP ανταμείβει τα συστήματα που δεν βρίσκουν απλώς σχετικά αντικείμενα, αλλά τα τοποθετούν και σε υψηλές θέσεις μέσα στη λίστα συστάσεων.

Η μετρική NDCG@K , δηλαδή *Normalized Discounted Cumulative Gain*, αξιολογεί επίσης την ποιότητα της κατάταξης, δίνοντας μεγαλύτερη βαρύτητα στα relevant αντικείμενα που εμφανίζονται στις πρώτες θέσεις. Αρχικά υπολογίζεται το DCG@K , το οποίο μπορεί να γραφεί ως:

$$DCG@K = \sum_{i=1}^K \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

όπου rel_i είναι ο βαθμός συνάφειας του αντικειμένου που βρίσκεται στη θέση i . Στην περίπτωση της παρούσας εργασίας, η συνάφεια αντιμετωπίζεται δυαδικά, δηλαδή ένα αντικείμενο είναι είτε relevant είτε μη relevant. Στη συνέχεια, το $DCG@K$ κανονικοποιείται με βάση το ιδανικό $IDCG@K$, δηλαδή την τιμή που θα είχε η τέλεια δυνατή κατάταξη:

$$NDCG_K = \frac{DCG_K}{IDCG_K}$$

Η τιμή της $NDCG@K$ κυμαίνεται από 0 έως 1, με την τιμή 1 να αντιστοιχεί σε ιδανική κατάταξη. Στην πράξη, η $NDCG@10$ δείχνει κατά πόσο οι relevant ταινίες εμφανίζονται όσο το δυνατόν πιο ψηλά στη λίστα των δέκα προτάσεων. Η μετρική αυτή είναι σημαντική επειδή σε ένα πραγματικό recommender system οι πρώτες θέσεις έχουν μεγαλύτερη σημασία, καθώς ο χρήστης είναι πιθανότερο να δει και να επιλέξει τα πρώτα αντικείμενα της λίστας (Jadon & Patil, 2024).

Το $HitRate@K$ χρησιμοποιείται στην παρούσα εργασία με τον δυαδικό ορισμό ανά χρήστη. Συγκεκριμένα, η μετρική λαμβάνει τιμή 1 όταν υπάρχει τουλάχιστον ένα relevant αντικείμενο μέσα στις πρώτες K προτάσεις και τιμή 0 όταν δεν υπάρχει κανένα. Ο τύπος είναι:

$$HitRate@K(u) = I(|rel(u) \cap rec_K(u)| > 0)$$

όπου $rel(u)$ είναι το σύνολο των σχετικών αντικειμένων για τον χρήστη u , $rec_K(u)$ είναι το σύνολο των K προτεινόμενων αντικειμένων και I είναι συνάρτηση ένδειξης που επιστρέφει 1 όταν η συνθήκη ισχύει και 0 διαφορετικά.

Η συνολική τιμή της μετρικής προκύπτει ως ο μέσος όρος των τιμών όλων των χρηστών:

$$HitRate@K = \frac{1}{|U|} \sum_{u \in U} HitRate@K(u)$$

Στον δεύτερο τύπο, το U συμβολίζει το σύνολο των χρηστών που συμμετέχουν στην αξιολόγηση, ενώ το $|U|$ εκφράζει το πλήθος αυτών των χρηστών. Ο όρος

$\sum_{u \in U} \text{HitRate}@K(u)$ αθροίζει τις επιμέρους τιμές HitRate όλων των χρηστών, δηλαδή προσθέτει 1 για κάθε χρήστη που είχε τουλάχιστον μία σχετική πρόταση στο top-K και 0 για κάθε χρήστη που δεν είχε καμία. Ο παράγοντας $\frac{1}{|U|}$ χρησιμοποιείται για να υπολογιστεί ο μέσος όρος αυτών των τιμών. Επομένως, η τελική τιμή του HitRate@K εκφράζει το ποσοστό των χρηστών για τους οποίους το σύστημα πέτυχε τουλάχιστον μία σχετική πρόταση μέσα στις πρώτες K θέσεις. Με αυτόν τον ορισμό, η τιμή της μετρικής κυμαίνεται πάντα από 0 έως 1. (Tamm et al., 2021)

Παράλληλα, χρησιμοποιείται και η μετρική Diversity, ώστε να εξεταστεί αν οι προτεινόμενες ταινίες παρουσιάζουν εσωτερική ποικιλία ή αν περιορίζονται σε πολύ παρόμοια αντικείμενα. Παρότι μεγάλο μέρος της έρευνας στα recommender systems έχει επικεντρωθεί στην ακρίβεια, υπάρχει αυξανόμενη συμφωνία ότι η accuracy από μόνη της δεν αρκεί για την αξιολόγηση της πρακτικής αποτελεσματικότητας ενός συστήματος. Επιπλέον, ορίζουν τη diversity ως ιδιότητα που αφορά ένα σύνολο αντικειμένων και σχετίζεται με το πόσο διαφορετικά είναι τα αντικείμενα μεταξύ τους. Το Diversity υπολογίζεται ως εξής:

$$\text{Diversity}(R) = 1 - \frac{1}{N} \sum_{i < j} \text{cosine_similarity}(i, j)$$

Όπου i και j είναι δύο διαφορετικές ταινίες που ανήκουν στη λίστα προτάσεων, ενώ η συνθήκη $i < j$ δηλώνει ότι κάθε ζεύγος ταινιών υπολογίζεται μόνο μία φορά και ότι δεν πραγματοποιείται σύγκριση μιας ταινίας με τον εαυτό της. Το N εκφράζει το συνολικό πλήθος των μοναδικών ζευγών ταινιών που σχηματίζονται μέσα στη λίστα προτάσεων. Αν η λίστα περιλαμβάνει R ταινίες, τότε το πλήθος των ζευγών είναι:

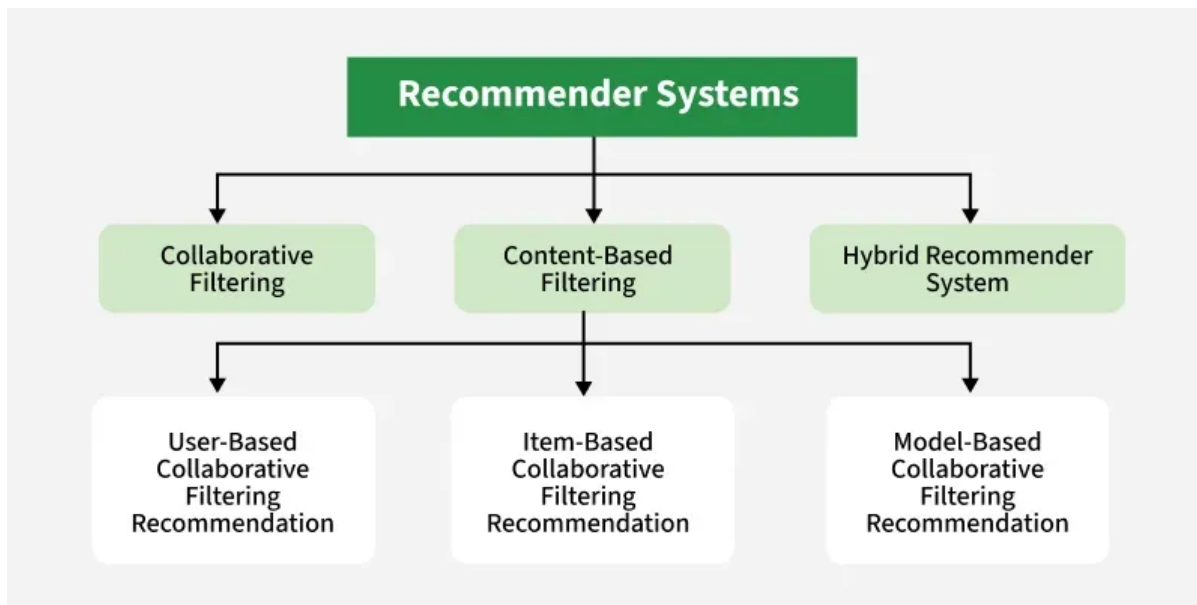
$$N = \frac{|R| (|R| - 1)}{2}$$

Στην παρούσα εργασία, όπου αξιολογούνται λίστες top-10 προτάσεων, το πλήθος των μοναδικών ζευγών είναι:

$$N = \frac{10 \cdot 9}{2} = 45$$

Η τιμή cosine similarity (i, j) εκφράζει τον βαθμό ομοιότητας ανάμεσα στις δύο ταινίες i και j . Επομένως, ο τύπος υπολογίζει αρχικά τον μέσο όρο της ομοιότητας όλων των ζευγών ταινιών της λίστας και στη συνέχεια τον αφαιρεί από το 1. Με αυτόν τον τρόπο, όσο πιο όμοιες είναι μεταξύ τους οι προτεινόμενες ταινίες, τόσο μικρότερη είναι η τιμή της Diversity. Αντίθετα, όσο περισσότερο διαφέρουν μεταξύ τους, τόσο μεγαλύτερη είναι η τιμή της μετρικής. (Mobasher, 2011)

4. Μοντέλα Συστήματος Προτάσεων



Εικόνα 21: Προσεγγίσεις Συστημάτων Προτάσεων, Πηγή: [geeksforgeeks.org](https://www.geeksforgeeks.org)

Για να μπορεί ένα σύστημα προτάσεων να παρέχει αξιόπιστες και στοχευμένες προτάσεις, απαιτείται η εφαρμογή μεθόδων που είναι ταυτόχρονα αποτελεσματικές και υψηλής ακρίβειας. Υπό αυτό το πρίσμα, είναι κρίσιμο να γνωρίζει κανείς πως λειτουργούν και τι μπορούν να προσφέρουν οι διάφορες τεχνικές που χρησιμοποιούνται στον χώρο των συστημάτων προτάσεων (Isinkaye et al., 2015).

Τα συστήματα προτάσεων μπορούν να κατηγοριοποιηθούν σε τρεις βασικές προσεγγίσεις. Το Content-Based Filtering, το Collaborative Filtering και το Hybrid Filtering. Στην Content-Based προσέγγιση, το σύστημα συλλέγει τα δεδομένα για την συμπεριφορά του χρήστη, είτε μέσω άμεσων ενδείξεων, όπως βαθμολογίες, είτε μέσω έμμεσων ενεργειών όπως κλικ και προβολές. Αυτή η πληροφορία χρησιμοποιείται για την δημιουργία ενός μοναδικού προφίλ χρήστη, το οποίο στη συνέχεια καθοδηγεί τις προτάσεις. Εδώ, κάθε πρόταση βασίζεται αποκλειστικά στην ιστορική δραστηριότητα του συγκεκριμένου χρήστη και επιχειρεί να προτείνει αντικείμενα που μοιάζουν με όσα έχει ήδη αξιολογήσει ή χρησιμοποιήσει.

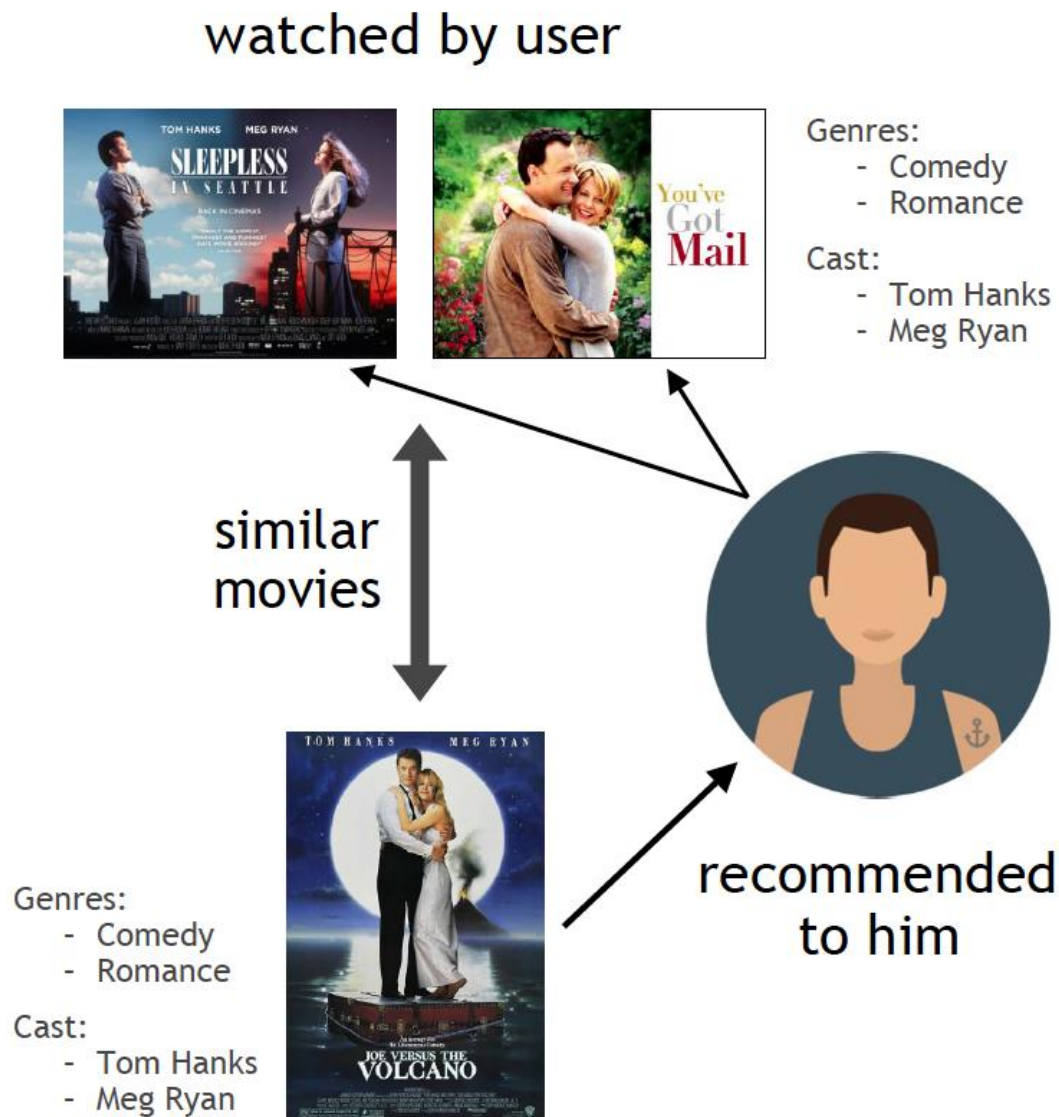
Σε αντίθεση με αυτό, το Collaborative Filtering εστιάζει στην σύγκριση των προτιμήσεων μεταξύ χρηστών. Το σύστημα εντοπίζει άτομα με παρόμοιο γούστο και προτείνει στον τρέχοντα χρήστη προϊόντα που άρεσαν στους ομοϊδεάτες του. Με αυτό τον τρόπο το

σύστημα αξιοποιεί την συλλογική εμπειρία και τις κοινές τάσεις για να υπολογίσει τις πιθανότητες ότι ένα προϊόν θα είναι επιθυμητό.

Καθώς και οι δύο μέθοδοι παρουσιάζουν περιορισμούς, για παράδειγμα, το Content-Based Filtering μπορεί να στερείται ποικιλίας, ενώ το Collaborative δυσκολεύεται με νέους χρήστες ή προϊόντα, οι υβριδικές προσεγγίσεις συνδυάζουν τα πλεονεκτήματα και των δύο. Οι Hybrid Filtering μέθοδοι μπορούν να λειτουργήσουν είτε με αλληλουχία εφαρμογής των δύο τεχνικών, είτε με ενσωμάτωση τους σε ένα κοινό μοντέλο, εξασφαλίζοντας πιο ολοκληρωμένες και αξιόπιστες προτάσεις. Επιπλέον, τέτοιες στρατηγικές είναι χρήσιμες για την αντιμετώπιση προκλήσεων όπως η αρχική έλλειψη δεδομένων (cold-start problem), η αραίωση του πίνακα αξιολογήσεων και η ανάγκη για κλιμάκωση σε μεγάλα datasets (Goyani & Chaurasiya, 2020).

4.1 Content-Based Filtering

Content-based Filtering



Εικόνα 22: Content Based Filtering, Πηγή: nvidia.com

Τα συστήματα προτάσεων που βασίζονται στο περιεχόμενο (content-based) αποτελούν μία κατηγορία τεχνικών όπου η διαδικασία παραγωγής προτάσεων βασίζεται αποκλειστικά στις ιδιότητες των αντικειμένων και στο προφίλ του εκάστοτε χρήστη. Η

προσέγγιση αυτή εστιάζει στην συσχέτιση μεταξύ των στοιχείων του ίδιου του αντικειμένου και του ιστορικού αλληλεπίδρασης του χρήστη. Με τον τρόπο αυτό, αποφεύγεται η εξάρτηση από δεδομένα άλλων χρηστών, αλλά και τα προβλήματα που προκύπτουν όταν άτομα με παρόμοια δημογραφικά χαρακτηριστικά καταλήγουν να έχουν εντελώς διαφορετικά γούστα.

Στην πράξη, οι content-based μέθοδοι δημιουργούν, αρχικά, προφίλ αντικειμένων, τα οποία περιγράφουν κάθε στοιχείο μέσω των χαρακτηριστικών του. Για παράδειγμα, σε ένα σύστημα για ταινίες μπορεί να χρησιμοποιηθούν παράμετροι όπως το είδος, οι ηθοποιοί, ο σκηνοθέτης ή ακόμη και λέξεις κλειδιά από την περίληψη της ταινίας. Επόμενο βήμα είναι η δημιουργία ενός προφίλ χρήστη, το οποίο προκύπτει από την συγκέντρωση των χαρακτηριστικών των αντικειμένων που ο χρήστης έχει αξιολογήσει θετικά. Στόχος είναι να αποτυπωθεί όσο το δυνατόν πιο ολοκληρωμένα το ύφος, οι προτιμήσεις, και οι θεματικές που αντανακλούν τα ενδιαφέροντά του. Το προφίλ αυτό συγκρίνεται με τα χαρακτηριστικά νέων αντικειμένων, και όσα παρουσιάζουν υψηλή ομοιότητα θεωρούνται κατάλληλα προς σύσταση.

Για να πραγματοποιηθεί αυτή η σύγκριση χρησιμοποιούνται διάφορες τεχνικές επεξεργασίας κειμένου, όπως η μετατροπή των λεκτικών περιγραφών σε αριθμητικά διανύσματα μέσω μεθόδων vectorization (πχ, TF-IDF). Στον πολυδιάστατο αυτό χώρο, κάθε ταινία μετατρέπεται σε ένα διάνυσμα χαρακτηριστικών. Η ομοιότητα μεταξύ δύο αντικειμένων υπολογίζεται, συχνά, με την συνάρτηση cosine similarity, η οποία μετρά την σχέση μεταξύ δύο διανυσμάτων σε τιμές που κυμαίνονται από -1 έως 1. Όσο πλησιέστερη είναι η τιμή της συνάρτησης στο 1, τόσο περισσότερο τα δύο αντικείμενα μοιράζονται κοινά γνωρίσματα, άρα θεωρείται πιθανότερο ότι θα ενδιαφέρουν τον ίδιο χρήστη. Έτσι το σύστημα μπορεί να προτείνει ταινίες ή προϊόντα με παρόμοια χαρακτηριστικά με εκείνα που ο χρήστης έχει δείξει ότι προτιμάει.

Ένα από τα πλεονεκτήματα των content based τεχνικών είναι ότι μπορούν να προσαρμοστούν γρήγορα στις αλλαγές των προτιμήσεων ενός χρήστη. Καθώς το προφίλ ενημερώνεται δυναμικά με βάση τις πρόσφατες αλληλεπιδράσεις, το σύστημα καταφέρνει να παραμένει ευθυγραμμισμένο με τις τρέχουσες τάσεις και ανάγκες του. Επίσης, επειδή κάθε προφίλ χρήστη είναι αυστηρά προσωποποιημένο και δεν απαιτεί δεδομένα άλλων χρηστών, η μέθοδος αυτή εισάγει λιγότερους κινδύνους σχετικά με την ιδιωτικότητα. Επιπλέον, υποστηρίζει αποτελεσματικά και το λεγόμενο cold-start πρόβλημα για νέα αντικείμενα. Εφόσον ένα αντικείμενο έχει αρκετά περιγραφικά χαρακτηριστικά, μπορεί να συσταθεί στον χρήστη ακόμη κι αν δεν έχει αξιολογηθεί ποτέ στο παρελθόν.

Παρά τα πλεονεκτήματα, οι μέθοδοι content-based αντιμετωπίζουν και συγκεκριμένους περιορισμούς. Μία από τις βασικότερες προκλήσεις είναι η ανάγκη για πλούσια και ακριβή μεταδεδομένα σχετικά με τα αντικείμενα. Όταν η περιγραφή τους είναι ελλιπής ή ασαφής, το σύστημα αδυνατεί να τα αναλύσει και να τα συσχετίσει σωστά. Επιπλέον, η συνεχής προσκόλληση στο υπάρχον προφίλ του χρήστη ενδέχεται να οδηγήσει σε φαινόμενα, όπου του σύστημα προτείνει αντικείμενα υπερβολικά όμοια με τις προηγούμενες επιλογές, αποτυγχάνοντας να επεκτείνει τον ορίζοντα του χρήστη σε νέες κατηγορίες ή διαφορετικά είδη περιεχομένου. Έτσι, ενώ ο χρήστης μπορεί να λαμβάνει εξαιρετικά στοχευμένες προτάσεις, στερείται της πιθανότητας να ανακαλύψει εναλλακτικά αντικείμενα που θα μπορούσαν, επίσης, να του αρέσουν.

Τέλος, με όλα τα πλεονεκτήματα και μειονεκτήματα τους, τα συστήματα content-based αποτελούν σημαντικό μέρος της σύγχρονης τεχνολογίας προτάσεων, ιδιαίτερα σε περιβάλλοντα υπηρεσιών streaming, πωλήσεων προϊόντων κ.α. Ο συνδυασμός της αυτονομίας τους από άλλα προφίλ χρηστών, και της δυνατότητας άμεσης προσαρμογής σε ατομικές προτιμήσεις, τα καθιστά ιδανική επιλογή για περιβάλλοντα όπου η εξατομίκευση και η ιδιωτικότητα αποτελούν καθοριστικούς παράγοντες (Goyani & Chaurasiya, 2020; Jayalakshmi et al., 2022; Kamble et al., 2025; Roy & Dutta, 2022).

4.1.1 Εμπλουτισμός Δεδομένων μέσω TMDb

Για την ενίσχυση του content-based μέρους του συστήματος σύστασης πραγματοποιήθηκε εμπλουτισμός των δεδομένων του MovieLens με πρόσθετα μεταδεδομένα από το TMDb. Το αρχικό MovieLens dataset περιλαμβάνει βασικές πληροφορίες για τις ταινίες, όπως τίτλο και είδη, όμως η πληροφορία αυτή είναι σχετικά περιορισμένη για τη δημιουργία ενός πιο εκφραστικού content profile. Για τον λόγο αυτό αξιοποιήθηκε το αρχείο links.csv, το οποίο συνδέει τα movieId του MovieLens με τα αντίστοιχα tmdbId, και μέσω του TMDb API ανακτήθηκαν επιπλέον χαρακτηριστικά, όπως περίληψη, tagline, λέξεις-κλειδιά, σκηνοθέτης, cast, εταιρείες παραγωγής, συλλογή, αρχικός τίτλος και TMDb genres. Τα δεδομένα αυτά αποθηκεύτηκαν στο αρχείο tmdb_enriched_movies.csv και χρησιμοποιήθηκαν στη συνέχεια για τη δημιουργία του εμπλουτισμένου πεδίου rich_content, το οποίο αποτέλεσε τη βάση της TF-IDF αναπαράστασης στο content-based και στο hybrid μοντέλο.

```
1 from pathlib import Path
2 import os
3 import time
4 import requests
5 import pandas as pd
6
7 # Paths
8 BASE_DIR = Path(__file__).resolve().parent.parent
9
10 LINKS_PATH = BASE_DIR / "data" / "ml-32m" / "links.csv"
11 TRAIN_PATH = BASE_DIR / "data" / "ml-32m" / "train.csv"
12 TEST_PATH = BASE_DIR / "data" / "ml-32m" / "test.csv"
13 OUTPUT_PATH = BASE_DIR / "data" / "ml-32m" / "tmdb_enriched_movies.csv"
14
15 # TMDb Auth
16 TMDB_BEARER_TOKEN = "MY_BEARER_TOKEN"
17
18 BASE_URL = "https://api.themoviedb.org/3"
19 HEADERS = {
20     "Authorization": f"Bearer {TMDB_BEARER_TOKEN}",
21     "accept": "application/json"
22 }
23
24 APPEND_TO_RESPONSE = "keywords,credits,external_ids"
```

Εικόνα 23: Ρύθμιση διαδρομών και σύνδεσης με το TMDB API

Στο αρχικό τμήμα του κώδικα εισάγονται οι απαραίτητες βιβλιοθήκες και ορίζονται οι διαδρομές των αρχείων. Η βιβλιοθήκη Path χρησιμοποιείται για τη δημιουργία σχετικών διαδρομών ανεξάρτητα από το σημείο εκτέλεσης του προγράμματος, ενώ οι os, time, requests και pandas χρησιμοποιούνται αντίστοιχα για βοηθητικές λειτουργίες, χρονική καθυστέρηση μεταξύ αιτημάτων, επικοινωνία με το TMDb API και διαχείριση των δεδομένων σε μορφή DataFrame. Το BASE_DIR ορίζεται ως ο βασικός φάκελος του project και στη συνέχεια δηλώνονται οι διαδρομές προς τα αρχεία links.csv, train.csv, test.csv και το τελικό αρχείο εξόδου tmdb_enriched_movies.csv.

Στη συνέχεια ορίζεται το κομμάτι αυθεντικοποίησης προς το TMDb API. Το TMDB_BEARER_TOKEN χρησιμοποιείται για την πρόσβαση στο API, ενώ το BASE_URL δείχνει στη βασική διεύθυνση του TMDb. Το λεξικό HEADERS περιλαμβάνει το authorization header και δηλώνει ότι η απάντηση αναμένεται σε μορφή JSON. Η μεταβλητή

APPEND_TO_RESPONSE = "keywords,credits,external_ids" είναι σημαντική, επειδή επιτρέπει στο ίδιο αίτημα προς το API να επιστραφούν επιπλέον πληροφορίες πέρα από τα βασικά στοιχεία της ταινίας, όπως λέξεις-κλειδιά, συντελεστές και εξωτερικά IDs. Με αυτόν τον τρόπο μειώνεται ο αριθμός των ξεχωριστών αιτημάτων που θα χρειαζόταν να γίνουν.

```

27 def safe_get_movie_details(tmdb_id): 1 usage
28     url = f"{BASE_URL}/movie/{tmdb_id}"
29     params = {
30         "append_to_response": APPEND_TO_RESPONSE,
31         "language": "en-US"
32     }
33
34     response = requests.get(url, headers=HEADERS, params=params, timeout=30)
35
36     if response.status_code == 401:
37         raise RuntimeError(f"401 Unauthorized: {response.text}")
38
39     if response.status_code == 404:
40         print(f"TMDB 404 for tmdb_id={tmdb_id}")
41         return None
42
43     response.raise_for_status()
44     return response.json()
45
46
47 def extract_director(credits_obj): 1 usage
48     crew = credits_obj.get("crew", []) if isinstance(credits_obj, dict) else []
49     directors = [p.get("name", "") for p in crew if p.get("job") == "Director"]
50     return " ".join([x for x in directors[:3] if x])
51
52 def extract_cast(credits_obj, top_n=8): 1 usage
53     cast = credits_obj.get("cast", []) if isinstance(credits_obj, dict) else []
54     names = [p.get("name", "") for p in cast[:top_n]]
55     return " ".join([x for x in names if x])

```

Εικόνα 24: Ανάκτηση μεταδεδομένων από το TMDB API

Ακολουθούν οι βοηθητικές συναρτήσεις εξαγωγής πληροφορίας από την απάντηση του TMDB. Η `safe_get_movie_details` δέχεται ένα `tmdb_id`, δημιουργεί το αντίστοιχο URL και στέλνει αίτημα στο API. Αν το API επιστρέψει σφάλμα 401, τότε σημαίνει ότι υπάρχει πρόβλημα με την αυθεντικοποίηση, ενώ αν επιστρέψει 404, η συγκεκριμένη ταινία δεν εντοπίζεται στο TMDB και η συνάρτηση επιστρέφει `None`. Σε διαφορετική περίπτωση, η απάντηση μετατρέπεται σε JSON και επιστρέφεται για περαιτέρω επεξεργασία. Οι υπόλοιπες βοηθητικές συναρτήσεις χρησιμοποιούνται για να εξάγουν συγκεκριμένα πεδία από το

JSON, όπως η `extract_director`, η οποία εντοπίζει μέχρι τρεις σκηνοθέτες από το `crew` και η `extract_cast` η οποία κρατά τα πρώτα ονόματα του `cast`.

```
57 def extract_keywords(keywords_obj): 1 usage
58     keywords = keywords_obj.get("keywords", []) if isinstance(keywords_obj, dict) else []
59     names = [k.get("name", "") for k in keywords]
60     return " ".join([x for x in names if x])
61
62 def extract_genre_names(genres_obj): 1 usage
63     if not isinstance(genres_obj, list):
64         return ""
65     return " ".join([g.get("name", "") for g in genres_obj if g.get("name")])
66
67 def extract_company_names(companies_obj): 1 usage
68     if not isinstance(companies_obj, list):
69         return ""
70     return " ".join([c.get("name", "") for c in companies_obj[:5] if c.get("name")])
71
72 def extract_collection_name(collection_obj): 1 usage
73     if isinstance(collection_obj, dict):
74         return collection_obj.get("name", "") or ""
75     return ""
```

Εικόνα 25: Εξαγωγή μεταδεδομένων ταινιών από το TMDB

Ο κώδικας συνεχίζει και στις υπόλοιπες συναρτήσεις, όπου έχουμε την `extract_keywords`, η οποία ενώνει τις λέξεις-κλειδιά, την `extract_genre_names` η οποία εξάγει τα `genres` του TMDB, την `extract_company_names` η οποία κρατά τις βασικές εταιρείες παραγωγής και την `extract_collection_name` η οποία επιστρέφει το όνομα της συλλογής αν η ταινία ανήκει σε κάποιο franchise ή collection.

```

77 # Main
78 def main():
79     print("BASE_DIR:", BASE_DIR)
80     print("LINKS_PATH exists:", LINKS_PATH.exists())
81     print("TRAIN_PATH exists:", TRAIN_PATH.exists())
82     print("TEST_PATH exists:", TEST_PATH.exists())
83     print("OUTPUT_PATH:", OUTPUT_PATH)
84
85     if not TMDB_BEARER_TOKEN:
86         raise ValueError("Δεν έχεις βάλει TMDB token στο environment variable TMDB_BEARER_TOKEN.")
87
88     if not LINKS_PATH.exists():
89         raise FileNotFoundError(f"Δεν βρέθηκε το links.csv: {LINKS_PATH}")
90     if not TRAIN_PATH.exists():
91         raise FileNotFoundError(f"Δεν βρέθηκε το train.csv: {TRAIN_PATH}")
92     if not TEST_PATH.exists():
93         raise FileNotFoundError(f"Δεν βρέθηκε το test.csv: {TEST_PATH}")
94
95     train = pd.read_csv(TRAIN_PATH, usecols=["movieId"])
96     test = pd.read_csv(TEST_PATH, usecols=["movieId"])
97
98     relevant_movie_ids = set(train["movieId"].unique()) | set(test["movieId"].unique())
99     print("Relevant unique movieIds from train/test:", len(relevant_movie_ids))
100
101     links = pd.read_csv(LINKS_PATH)
102
103     if "tmdbId" not in links.columns:
104         raise ValueError("To links.csv δεν περιέχει στήλη tmdbId")

```

Εικόνα 26: Έλεγχος αρχείων και επιλογή ταινιών για TMDB εμπλουτισμό

Στο κύριο μέρος του προγράμματος, η συνάρτηση main ξεκινά με εκτυπώσεις ελέγχου για τις διαδρομές των αρχείων. Με αυτόν τον τρόπο επιβεβαιώνεται ότι τα απαραίτητα αρχεία υπάρχουν στη σωστή θέση πριν ξεκινήσει η διαδικασία εμπλουτισμού. Στη συνέχεια γίνεται έλεγχος για την ύπαρξη του TMDB token και των αρχείων links.csv, train.csv και test.csv. Αν κάποιο από αυτά λείπει, το πρόγραμμα σταματά με κατάλληλο μήνυμα σφάλματος. Η λογική αυτή είναι χρήσιμη, επειδή αποτρέπει την εκτέλεση του script με ελλιπή δεδομένα.

Έπειτα φορτώνονται μόνο οι στήλες movieId από τα train.csv και test.csv. Από αυτά τα δύο αρχεία δημιουργείται το σύνολο relevant_movie_ids, το οποίο περιλαμβάνει όλες τις μοναδικές ταινίες που εμφανίζονται είτε στο train είτε στο test set. Η επιλογή αυτή έγινε ώστε να εμπλουτιστούν μόνο οι ταινίες που είναι πραγματικά χρήσιμες για τα μοντέλα και όχι όλες οι ταινίες του αρχικού movies.csv. Στη συγκεκριμένη εκτέλεση, το σύνολο αυτό

περιλαμβάνει 31.961 μοναδικά movieId, δηλαδή τις ταινίες που εμφανίζονται στα δεδομένα αξιολογήσεων μετά το split.

Στη συνέχεια φορτώνεται το links.csv, το οποίο περιέχει τις αντιστοιχίσεις ανάμεσα στα movieId του MovieLens και στα αντίστοιχα tmdbId. Ο κώδικας ελέγχει αν υπάρχει η στήλη tmdbId, κρατά μόνο τις ταινίες που ανήκουν στα relevant_movie_ids και αφαιρεί όσες δεν έχουν διαθέσιμο tmdbId.

```

106     links = links[links["movieId"].isin(relevant_movie_ids)].copy()
107     links = links[links["tmdbId"].notna()].copy()
108     links["tmdbId"] = links["tmdbId"].astype(int)
109     print("Movies to enrich after filtering:", len(links))
110
111     already_done = set()
112     if OUTPUT_PATH.exists():
113         try:
114             existing = pd.read_csv(OUTPUT_PATH, usecols=["movieId"])
115             already_done = set(existing["movieId"].unique())
116             print("Already enriched movieIds found:", len(already_done))
117         except Exception:
118             print("Existing output found but could not be read cleanly.")
119
120     links_to_process = links[~links["movieId"].isin(already_done)].copy()
121     print("Remaining movies to process:", len(links_to_process))
122
123     rows = []
124     total = len(links_to_process)
125     for i, row in enumerate(links_to_process.itertuples(index=False), start=1):
126         movie_id = row.movieId
127         tmdb_id = row.tmdbId
128         imdb_id = getattr(row, "imdbId", None)
129         try:
130             data = safe_get_movie_details(tmdb_id)
131         except Exception as e:
132             print(f"Error for movieId={movie_id}, tmdb_id={tmdb_id}: {e}")
133             continue

```

Εικόνα 27: Φιλτράρισμα και έλεγχος ταινιών προς εμπλουτισμό

Σε αυτό το τμήμα ο κώδικας προετοιμάζει τις ταινίες που θα εμπλουτιστούν μέσω του TMDb API και ξεκινά τη διαδικασία αποστολής αιτημάτων. Αρχικά, ο πίνακας links φιλτράρεται ώστε να κρατηθούν μόνο οι ταινίες των οποίων το movieId υπάρχει στο σύνολο relevant_movie_ids. Το σύνολο αυτό έχει δημιουργηθεί από τα train και test αρχεία, άρα περιλαμβάνει μόνο τις ταινίες που χρησιμοποιούνται πραγματικά στην εργασία. Με αυτόν

τον τρόπο, ο εμπλουτισμός δεν γίνεται για όλες τις ταινίες του αρχικού MovieLens dataset, αλλά μόνο για όσες είναι χρήσιμες για τα μοντέλα σύστασης.

Στη συνέχεια αφαιρούνται οι ταινίες που δεν έχουν διαθέσιμο `tmdbId`, επειδή το συγκεκριμένο πεδίο είναι απαραίτητο για την αναζήτηση της ταινίας στο TMDb. Αμέσως μετά, η στήλη `tmdbId` μετατρέπεται σε ακέραιο αριθμό με την `astype(int)`, ώστε το ID να μπορεί να χρησιμοποιηθεί σωστά στο URL του API. Η εκτύπωση `Movies to enrich after filtering` εμφανίζει πόσες ταινίες απομένουν μετά από αυτό το φιλτράρισμα και μπορούν θεωρητικά να εμπλουτιστούν.

Επιπλέον, ο κώδικας ελέγχει αν υπάρχει ήδη το αρχείο εξόδου `tmdb_enriched_movies.csv`. Αν υπάρχει, διαβάζει μόνο τη στήλη `movieId` και δημιουργεί το σύνολο `already_done`, το οποίο περιλαμβάνει τις ταινίες που έχουν ήδη εμπλουτιστεί σε προηγούμενη εκτέλεση. Αυτό είναι σημαντικό, επειδή η διαδικασία εμπλουτισμού μέσω API μπορεί να είναι χρονοβόρα. Με αυτόν τον μηχανισμό, το script δεν επαναλαμβάνει αιτήματα για ταινίες που έχουν ήδη αποθηκευτεί, αλλά συνεχίζει μόνο με όσες λείπουν. Αν το υπάρχον αρχείο δεν μπορεί να διαβαστεί σωστά, εμφανίζεται σχετικό μήνυμα, χωρίς να διακόπτεται άμεσα η εκτέλεση.

Στο επόμενο στάδιο δημιουργείται το `links_to_process`, το οποίο περιλαμβάνει μόνο τις ταινίες που δεν υπάρχουν στο `already_done`. Δηλαδή, από τις ταινίες που έχουν έγκυρο `tmdbId`, αφαιρούνται όσες έχουν ήδη εμπλουτιστεί. Η εκτύπωση `Remaining movies to process` δείχνει πόσες ταινίες απομένουν πραγματικά για νέα επεξεργασία. Έτσι, ο κώδικας είναι επαναλήψιμος και αποδοτικός, καθώς μπορεί να εκτελεστεί πολλές φορές χωρίς να ξεκινά κάθε φορά από την αρχή.

Αφού καθοριστούν οι ταινίες που απομένουν, αρχικοποιείται η λίστα `rows`, στην οποία θα αποθηκευτούν προσωρινά οι νέες εμπλουτισμένες εγγραφές. Η μεταβλητή `total` κρατά το πλήθος των ταινιών που πρόκειται να επεξεργαστούν. Έπειτα, το `for loop` διατρέχει κάθε γραμμή του `links_to_process`. Για κάθε ταινία εξάγεται το `movie_id`, το αντίστοιχο `tmdb_id` και, αν υπάρχει, το `imdb_id`. Η χρήση του `getattr(row, "imdbId", None)` εξασφαλίζει ότι, ακόμη και αν δεν υπάρχει διαθέσιμο `imdbId`, ο κώδικας δεν θα διακοπεί αλλά θα συνεχίσει με τιμή `None`.

Τελικώς, μέσα στο `try block` καλείται η συνάρτηση `safe_get_movie_details(tmdb_id)`, η οποία στέλνει το αίτημα στο TMDb API για τη συγκεκριμένη ταινία. Αν κατά την κλήση προκύψει οποιοδήποτε σφάλμα, αυτό αποθηκεύεται στη μεταβλητή `e`, εμφανίζεται μήνυμα

που περιλαμβάνει το `movieId`, το `tmdb_id` και την αιτία του σφάλματος, και στη συνέχεια η εντολή `continue` μεταφέρει την εκτέλεση στην επόμενη ταινία. Με αυτόν τον τρόπο, ένα προβληματικό API request δεν σταματά ολόκληρη τη διαδικασία εμπλουτισμού, αλλά απλώς παραλείπεται η συγκεκριμένη ταινία.

Παρακάτω, μέσα στο `try block` καλείται η συνάρτηση `safe_get_movie_details(tmdb_id)`, η οποία στέλνει το αίτημα στο TMDb API για την ανάκτηση των πρόσθετων πληροφοριών της συγκεκριμένης ταινίας. Αν το αίτημα ολοκληρωθεί επιτυχώς, τα δεδομένα που επιστρέφονται αποθηκεύονται προσωρινά στη μεταβλητή `data`, ώστε να χρησιμοποιηθούν στα επόμενα βήματα του εμπλουτισμού. Αν προκύψει κάποιο σφάλμα κατά την κλήση του API, για παράδειγμα λόγω μη διαθέσιμης ταινίας, προβλήματος σύνδεσης ή άλλης αποτυχίας του request, το σφάλμα καταγράφεται με μήνυμα που περιλαμβάνει το `movieId` και το `tmdb_id`. Έπειτα, με την εντολή `continue`, ο κώδικας παραλείπει τη συγκεκριμένη ταινία και συνεχίζει με την επόμενη, χωρίς να διακόπτεται ολόκληρη η διαδικασία εμπλουτισμού.

```
135         if data is None:
136             continue
137
138         enriched = {
139             "movieId": movie_id,
140             "tmdbId": tmdb_id,
141             "imdbId_from_links": imdb_id,
142             "tmdb_title": data.get("title", ""),
143             "original_title": data.get("original_title", ""),
144             "overview": data.get("overview", ""),
145             "tagline": data.get("tagline", ""),
146             "release_date": data.get("release_date", ""),
147             "runtime": data.get("runtime", None),
148             "vote_average": data.get("vote_average", None),
149             "vote_count": data.get("vote_count", None),
150             "popularity": data.get("popularity", None),
151             "status": data.get("status", ""),
152             "original_language": data.get("original_language", ""),
153             "genres_tmdb": extract_genre_names(data.get("genres", [])),
154             "keywords_tmdb": extract_keywords(data.get("keywords", {})),
155             "director": extract_director(data.get("credits", {})),
156             "cast": extract_cast(data.get("credits", {}), top_n=8),
157             "production_companies": extract_company_names(data.get("production_companies", [])),
158             "collection": extract_collection_name(data.get("belongs_to_collection")),
159             "imdb_id_tmdb": (data.get("external_ids", {}) or {}).get("imdb_id", "")
160         }
161
162         rows.append(enriched)
```

Εικόνα 28: Δημιουργία εμπλουτισμένων μεταδεδομένων ταινίας

Σε αυτό το τμήμα ο κώδικας επεξεργάζεται την απάντηση που έχει επιστραφεί από το TMDb API και δημιουργεί την τελική εμπλουτισμένη εγγραφή για κάθε ταινία. Αρχικά γίνεται έλεγχος αν η μεταβλητή data είναι None. Αυτό μπορεί να συμβεί, για παράδειγμα, όταν η συνάρτηση `safe_get_movie_details` επιστρέψει None λόγω σφάλματος 404, δηλαδή όταν η συγκεκριμένη ταινία δεν βρεθεί στο TMDb. Σε αυτή την περίπτωση, η ταινία παραλείπεται με `continue` και η διαδικασία συνεχίζει με την επόμενη εγγραφή.

Αν υπάρχουν διαθέσιμα δεδομένα, δημιουργείται το λεξικό `enriched`, το οποίο συγκεντρώνει όλες τις πληροφορίες που θα αποθηκευτούν στο τελικό αρχείο `tmdb_enriched_movies.csv`. Αρχικά, αποθηκεύονται τα βασικά αναγνωριστικά της ταινίας, δηλαδή το `movieId` του MovieLens, το `tmdbId` του TMDb και το `imdbId_from_links` που προέρχεται από το αρχείο `links.csv` και καταγράφονται βασικά πεδία από το TMDb, όπως ο τίτλος της ταινίας, ο αρχικός τίτλος, η περίληψη, το `tagline`, η ημερομηνία κυκλοφορίας, η διάρκεια, η μέση βαθμολογία, το πλήθος ψήφων, η δημοτικότητα, η κατάσταση κυκλοφορίας και η αρχική γλώσσα.

Ιδιαίτερη σημασία έχουν τα πεδία που χρησιμοποιούνται αργότερα για την ενίσχυση του content-based μοντέλου. Η στήλη `genres_tmdb` προκύπτει από τη συνάρτηση `extract_genre_names`, η οποία εξάγει τα είδη της ταινίας από τα δεδομένα του TMDb. Η στήλη `keywords_tmdb` δημιουργείται από τη συνάρτηση `extract_keywords` και περιλαμβάνει λέξεις-κλειδιά που περιγράφουν θεματικά την ταινία. Η στήλη `director` εξάγει τον ή τους σκηνοθέτες από τα `credits`, ενώ η στήλη `cast` κρατά τα πρώτα ονόματα του `cast`, με όριο τα 8 άτομα. Επιπλέον, η στήλη `production_companies` αποθηκεύει βασικές εταιρείες παραγωγής, ενώ η στήλη `collection` κρατά το όνομα της συλλογής στην οποία μπορεί να ανήκει μια ταινία, για παράδειγμα κάποιο franchise ή σειρά ταινιών.

Τέλος, αποθηκεύεται και το `imdb_id_tmdb`, το οποίο προέρχεται από τα `external_ids` του TMDb. Η χρήση της έκφρασης `(data.get("external_ids", {}) or {}).get("imdb_id", "")` εξασφαλίζει ότι αν τα `external_ids` λείπουν ή είναι κενά, δεν θα προκληθεί σφάλμα και θα επιστραφεί απλώς κενό string. Αφού δημιουργηθεί το λεξικό `enriched`, προστίθεται στη λίστα `rows`. Η λίστα αυτή λειτουργεί ως προσωρινός χώρος αποθήκευσης όλων των νέων εμπλουτισμένων εγγραφών, οι οποίες στο τέλος της διαδικασίας θα μετατραπούν σε `DataFrame` και θα αποθηκευτούν στο αρχείο εξόδου.

```

164         if i % 100 == 0 or i == total:
165             print(f"Processed {i}/{total}")
166
167             time.sleep(0.05)
168
169         new_df = pd.DataFrame(rows)
170
171         if OUTPUT_PATH.exists():
172             try:
173                 old_df = pd.read_csv(OUTPUT_PATH)
174                 final_df = pd.concat(objs=[old_df, new_df], ignore_index=True)
175                 final_df = final_df.drop_duplicates(subset=["movieId"], keep="last")
176             except Exception:
177                 final_df = new_df.copy()
178         else:
179             final_df = new_df.copy()
180
181         final_df.to_csv(OUTPUT_PATH, index=False, encoding="utf-8-sig")
182         print(f"Saved: {OUTPUT_PATH}")
183         print("Final shape:", final_df.shape)
184
185
186  ► if __name__ == "__main__":
187         main()

```

Εικόνα 29: Αποθήκευση εμπλουτισμένων δεδομένων TMDb

Σε αυτό το τελευταίο τμήμα του κώδικα ολοκληρώνεται η διαδικασία εμπλουτισμού και αποθήκευσης των δεδομένων. Μέσα στο for loop, εμφανίζεται μήνυμα προόδου κάθε 100 ταινίες ή όταν η διαδικασία φτάσει στην τελευταία ταινία προς επεξεργασία. Αυτό γίνεται με τη συνθήκη `if i % 100 == 0 or i == total`, όπου η μεταβλητή `i` δείχνει πόσες ταινίες έχουν ήδη επεξεργαστεί και η μεταβλητή `total` δείχνει το συνολικό πλήθος των ταινιών που πρόκειται να επεξεργαστούν. Η εκτύπωση αυτή είναι χρήσιμη, επειδή η ανάκτηση δεδομένων από το TMDb API μπορεί να διαρκέσει αρκετή ώρα, ιδιαίτερα όταν ο αριθμός των ταινιών είναι μεγάλος. Αμέσως μετά, η εντολή `time.sleep(0.05)` εισάγει μια μικρή καθυστέρηση ανάμεσα στα διαδοχικά αιτήματα προς το API. Με αυτόν τον τρόπο αποφεύγεται η αποστολή υπερβολικά πολλών requests σε πολύ μικρό χρονικό διάστημα και η διαδικασία γίνεται πιο σταθερή.

Αφού ολοκληρωθεί η επανάληψη, η λίστα `rows`, η οποία περιέχει τις νέες εμπλουτισμένες εγγραφές που ανακτήθηκαν στη συγκεκριμένη εκτέλεση, μετατρέπεται σε `DataFrame` με την εντολή `pd.DataFrame(rows)`. Το νέο αυτό `DataFrame` περιλαμβάνει μόνο

τις ταινίες που επεξεργάστηκε το script στην τρέχουσα εκτέλεση. Στη συνέχεια, ο κώδικας ελέγχει αν υπάρχει ήδη το αρχείο εξόδου `tmdb_enriched_movies.csv`. Αν το αρχείο υπάρχει, διαβάζεται ως `old_df` και ενώνεται με το νέο `DataFrame` μέσω της `pd.concat`. Έτσι, τα νέα δεδομένα προστίθενται στα ήδη αποθηκευμένα δεδομένα, χωρίς να χάνεται ο προηγούμενος εμπλουτισμός. Μετά τη συγχώνευση, εφαρμόζεται η `drop_duplicates` με βάση το `movieId`, κρατώντας την τελευταία εγγραφή σε περίπτωση που κάποια ταινία εμφανίζεται περισσότερες από μία φορές. Αυτό διασφαλίζει ότι στο τελικό αρχείο κάθε ταινία υπάρχει μόνο μία φορά και ότι, αν κάποια εγγραφή έχει ενημερωθεί σε νεότερη εκτέλεση, διατηρείται η πιο πρόσφατη εκδοχή της.

Αν το υπάρχον αρχείο εξόδου δεν μπορεί να διαβαστεί σωστά, ο κώδικας χρησιμοποιεί μόνο το νέο `DataFrame` της τρέχουσας εκτέλεσης, ώστε να μη διακοπεί η διαδικασία αποθήκευσης. Αν δεν υπάρχει καθόλου προηγούμενο αρχείο, τότε το τελικό `DataFrame` δημιουργείται επίσης αποκλειστικά από τα νέα δεδομένα. Τέλος, το `final_df` αποθηκεύεται στο `OUTPUT_PATH` με τη μορφή `CSV`, χωρίς να αποθηκευτεί ο `index` του `DataFrame` ως ξεχωριστή στήλη. Η κωδικοποίηση `utf-8-sig` χρησιμοποιείται για καλύτερη συμβατότητα με προγράμματα όπως το Excel, ειδικά όταν υπάρχουν ειδικοί χαρακτήρες ή μη αγγλικό κείμενο. Οι τελευταίες εκτυπώσεις εμφανίζουν τη διαδρομή στην οποία αποθηκεύτηκε το αρχείο και το τελικό του μέγεθος μέσω του `final_df.shape`. Η συνθήκη `if __name__ == "__main__": main()` δηλώνει ότι η συνάρτηση `main()` θα εκτελεστεί μόνο όταν το αρχείο τρέχει απευθείας ως κύριο πρόγραμμα. Συνολικά, το τμήμα αυτό διασφαλίζει ότι τα εμπλουτισμένα δεδομένα συλλέγονται, ενώνονται με τα ήδη υπάρχοντα, καθαρίζονται από διπλότυπα και αποθηκεύονται στο τελικό αρχείο `tmdb_enriched_movies.csv`, το οποίο χρησιμοποιείται στη συνέχεια για την ενίσχυση του `content-based` και του `hybrid` μοντέλου.

4.1.2 Ανάλυση αρχείου content-based.py

```
1 import re
2 from pathlib import Path
3 import numpy as np
4 import pandas as pd
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.metrics.pairwise import cosine_similarity
7 from sklearn.metrics import ndcg_score
8
9 # =====
10 # SETTINGS
11 # =====
12 TOP_K = 10
13 EVAL_SAMPLE_USERS = 300
14
15 PROFILE_THRESHOLD = 4.0
16 EVAL_THRESHOLD = 3.5
17
18 WEIGHT_MODE = "discrete-strong"
19
20 USE_POPULARITY = True
21 POPULARITY_WEIGHT = 0.05
22
23 BASE_DIR = Path(__file__).resolve().parent.parent
24
25 movies_path = BASE_DIR / "data" / "ml-32m" / "movies.csv"
26 tags_path = BASE_DIR / "data" / "ml-32m" / "tags.csv"
27 train_path = BASE_DIR / "data" / "ml-32m" / "train.csv"
28 test_path = BASE_DIR / "data" / "ml-32m" / "test.csv"
29 tmdb_path = BASE_DIR / "data" / "ml-32m" / "tmdb_enriched_movies.csv"
30
```

Εικόνα 30: Ρυθμίσεις και διαδρομές του συστήματος συστάσεων

Στο πρώτο σκέλος του κώδικα εισάγονται οι βασικές βιβλιοθήκες που υποστηρίζουν την προεπεξεργασία του κειμένου, τη διαχείριση των δεδομένων, τη διανυσματική αναπαράσταση του περιεχομένου και την αξιολόγηση του συστήματος. Η βιβλιοθήκη `re` χρησιμοποιείται για καθαρισμό κειμενικών πεδίων μέσω κανονικών εκφράσεων, ενώ η `Path` από τη `pathlib` αξιοποιείται για τον δυναμικό ορισμό των διαδρομών των αρχείων του project.

Οι βιβλιοθήκες `numpy` και `pandas` καλύπτουν τις αριθμητικές πράξεις και τη διαχείριση των πινάκων δεδομένων αντίστοιχα. Από το `scikit-learn` χρησιμοποιούνται το `TfidfVectorizer` για τη μετατροπή του περιεχομένου των ταινιών σε TF-IDF διανύσματα, η `cosine_similarity` για τον υπολογισμό της ομοιότητας μεταξύ του προφίλ του χρήστη και των ταινιών, και η `ndcg_score` για τον υπολογισμό της μετρικής NDCG κατά την αξιολόγηση.

Στη συνέχεια ορίζονται οι βασικές παράμετροι του συστήματος. Η μεταβλητή `TOP_K = 10` καθορίζει ότι το μοντέλο θα επιστρέφει δέκα προτάσεις ανά χρήστη, ενώ η `EVAL_SAMPLE_USERS = 300` ορίζει ότι η αξιολόγηση θα πραγματοποιείται σε δείγμα 300 χρηστών. Η παράμετρος `PROFILE_THRESHOLD = 4.0` προσδιορίζει ποιες αξιολογήσεις θεωρούνται αρκετά θετικές ώστε να συμμετέχουν στη δημιουργία του προφίλ του χρήστη, ενώ η `EVAL_THRESHOLD = 3.5` χρησιμοποιείται για τον ορισμό των *relevant* αντικειμένων στο *test set*. Η μεταβλητή `WEIGHT_MODE = "discrete_strong"` δηλώνει ότι οι υψηλότερες βαθμολογίες αποκτούν ισχυρότερη βαρύτητα κατά τη σύνθεση του *user profile*, ενισχύοντας περισσότερο τα χαρακτηριστικά των ταινιών που ο χρήστης έχει αξιολογήσει εντονότερα θετικά.

Παράλληλα, ενεργοποιείται η ενσωμάτωση της δημοφιλίας μέσω των παραμέτρων `USE_POPULARITY = True` και `POPULARITY_WEIGHT = 0.05`. Αυτό σημαίνει ότι το τελικό σκορ των ταινιών δεν βασίζεται αποκλειστικά στην ομοιότητα περιεχομένου, αλλά εμπλουτίζεται σε μικρό βαθμό και με πληροφορία δημοφιλίας από το *training set*. Έτσι, το μοντέλο παραμένει κατά βάση *content-based*, αλλά αποκτά και έναν περιορισμένο μηχανισμό εξομάλυνσης υπέρ πιο δημοφιλών τίτλων.

Τέλος, ορίζεται η βασική διαδρομή του *project* μέσω της εντολής `BASE_DIR = Path(__file__).resolve().parent.parent`, και πάνω σε αυτήν κατασκευάζονται οι διαδρομές προς τα αρχεία `movies.csv`, `tags.csv`, `train.csv`, `test.csv` και `tmdb_enriched_movies.csv`. Τα αρχεία αυτά αποτελούν τις κύριες πηγές πληροφορίας του συστήματος, καθώς περιλαμβάνουν τα βασικά μεταδεδομένα των ταινιών, τα *tags* των χρηστών, τα σύνολα εκπαίδευσης και αξιολόγησης, καθώς και τα εμπλουτισμένα χαρακτηριστικά από το TMDb. Με αυτό το στάδιο ολοκληρώνεται η αρχική παραμετροποίηση του συστήματος και τίθεται το πλαίσιο πάνω στο οποίο αναπτύσσεται ο υπόλοιπος recommender.

```

34 def clean_text(text): 4 usages new *
35     text = str(text).lower().strip()
36     text = re.sub(pattern: r"[^a-z0-9\s]", repl: " ", text)
37     text = re.sub(pattern: r"\s+", repl: " ", text).strip()
38     return text
39
40 def truncate_words(text, max_words=60): 1 usage new *
41     words = str(text).split()
42     return " ".join(words[:max_words])
43
44 def get_rating_weights(ratings_array, mode="discrete"): 1 usage
45     if mode == "linear":
46         weights = ratings_array - PROFILE_THRESHOLD + 1.0
47     elif mode == "raw":
48         weights = ratings_array
49     elif mode == "squared":
50         weights = (ratings_array - 3.5) ** 2
51     elif mode == "discrete":
52         weights = np.where(
53             ratings_array >= 5.0, 3.0,
54             np.where(ratings_array >= 4.5, 2.0, 1.0)
55         )
56     elif mode == "discrete_strong":
57         weights = np.where(
58             ratings_array >= 5.0, 4.0,
59             np.where(ratings_array >= 4.5, 2.5, 1.0)
60         )
61     else:
62         raise ValueError(f"Unknown WEIGHT_MODE: {mode}")
63
64     return weights.reshape(-1, 1)

```

Εικόνα 31: Καθαρισμός κειμένου και υπολογισμός βαρών αξιολογήσεων

Στο επόμενο τμήμα του κώδικα ορίζονται τρεις βοηθητικές συναρτήσεις που χρησιμοποιούνται στην προεπεξεργασία του περιεχομένου και στη διαμόρφωση του προφίλ του χρήστη. Η πρώτη συνάρτηση, `clean_text`, αναλαμβάνει τον καθαρισμό του κειμένου πριν αυτό αξιοποιηθεί στη δημιουργία των περιγραφικών χαρακτηριστικών των ταινιών. Σε πρώτη

φάση, το εισερχόμενο πεδίο μετατρέπεται σε συμβολοσειρά, ώστε να μπορεί να υποβληθεί σε ενιαία επεξεργασία ανεξάρτητα από την αρχική του μορφή και μετατρέπεται σε πεζά γράμματα και αφαιρούνται τα περιττά κενά στην αρχή και στο τέλος. Έπειτα, με χρήση κανονικής έκφρασης, κάθε χαρακτήρας που δεν είναι γράμμα, αριθμός ή κενό αντικαθίσταται με κενό διάστημα, γεγονός που απομακρύνει σημεία στίξης και άλλους ειδικούς χαρακτήρες. Ακολουθεί δεύτερος καθαρισμός, κατά τον οποίο πολλαπλά συνεχόμενα κενά συγχωνεύονται σε ένα μόνο, ώστε το τελικό κείμενο να είναι ομοιόμορφο και κατάλληλο για μετατροπή σε TF-IDF χαρακτηριστικά. Με αυτόν τον τρόπο, η συνάρτηση εξασφαλίζει ότι διαφορετικές μορφές γραφής ή θορυβώδη στοιχεία δεν επηρεάζουν αρνητικά τη διανυσματική αναπαράσταση του περιεχομένου.

Η δεύτερη συνάρτηση, `truncate_words`, χρησιμοποιείται για τον περιορισμό του μήκους μεγάλων κειμενικών πεδίων. Ο κώδικας μετατρέπει αρχικά το περιεχόμενο σε συμβολοσειρά, το διασπά σε επιμέρους λέξεις και στη συνέχεια διατηρεί μόνο τις πρώτες `max_words` λέξεις, οι οποίες ενώνονται ξανά σε ενιαίο κείμενο. Στην παρούσα υλοποίηση η συνάρτηση αυτή αξιοποιείται ώστε μεγάλα περιγραφικά πεδία, όπως το `overview`, να μην κυριαρχούν δυσανάλογα στη σύνθεση του περιεχομένου κάθε ταινίας.

Η τρίτη συνάρτηση, `get_rating_weights`, μετατρέπει τις βαθμολογίες των ταινιών σε αριθμητικά βάρη που χρησιμοποιούνται κατά τη δημιουργία του προφίλ του χρήστη. Η συνάρτηση υποστηρίζει διαφορετικά σχήματα στάθμισης, όπως `linear`, `raw`, `squared`, `discrete` και `discrete_strong`, επιτρέποντας πειραματισμό με εναλλακτικούς τρόπους ενίσχυσης των θετικών αξιολογήσεων. Στη συγκεκριμένη υλοποίηση, όμως, έχει επιλεγεί η ρύθμιση `WEIGHT_MODE = "discrete_strong"`, άρα αυτό είναι το σχήμα που εφαρμόζεται τελικά. Με βάση αυτό, οι ταινίες με βαθμολογία 5.0 λαμβάνουν βάρος 4.0, οι ταινίες με βαθμολογία από 4.5 έως μικρότερη του 5.0 λαμβάνουν βάρος 2.5, ενώ οι υπόλοιπες θετικές αξιολογήσεις λαμβάνουν βάρος 1.0. Έτσι, οι πιο υψηλά αξιολογημένες ταινίες επηρεάζουν πολύ περισσότερο το τελικό προφίλ του χρήστη από ό,τι οι απλώς θετικά αξιολογημένες. Αν δοθεί μη αποδεκτή τιμή στη μεταβλητή `mode`, η συνάρτηση επιστρέφει σφάλμα, αποτρέποντας τη χρήση άγνωστου σχήματος στάθμισης. Τέλος, τα βάρη επιστρέφονται σε μορφή στήλης με τη χρήση της `reshape(-1, 1)`, ώστε να μπορούν να χρησιμοποιηθούν σωστά στον πολλαπλασιασμό με τα διανύσματα χαρακτηριστικών των ταινιών.

```

66 def precision_at_k(recommended_items, relevant_items, k): 1usage new*
67     hits = sum(1 for item in recommended_items[:k] if item in relevant_items)
68     return hits / k if k > 0 else 0.0
69
70 def recall_at_k(recommended_items, relevant_items, k): 1usage new*
71     if len(relevant_items) == 0:
72         return 0.0
73     hits = sum(1 for item in recommended_items[:k] if item in relevant_items)
74     return hits / len(relevant_items)
75
76 def average_precision_at_k(recommended_items, relevant_items, k): 1usage new*
77     hits = 0
78     precision_sum = 0.0
79     for i, item in enumerate(recommended_items[:k], start=1):
80         if item in relevant_items:
81             hits += 1
82             precision_sum += hits / i
83     denom = min(len(relevant_items), k)
84     return precision_sum / denom if denom > 0 else 0.0
85
86 def ndcg_at_k(recommended_items, scores, relevant_items, k): 1usage new*
87     y_true = np.array([1 if item in relevant_items else 0 for item in recommended_items[:k]])
88     y_score = np.array([scores[:k]])
89     return ndcg_score(y_true, y_score)
90
91 def hit_rate_at_k(recommended_items, relevant_items, k): 1usage new*
92     return 1.0 if any(item in relevant_items for item in recommended_items[:k]) else 0.0

```

Εικόνα 32: Υλοποίηση μετρικών αξιολόγησης για τις προτεινόμενες ταινίες

Σε αυτό το κομμάτι ορίζονται οι βασικές συναρτήσεις αξιολόγησης του συστήματος προτάσεων, οι οποίες υπολογίζουν την ποιότητα των αποτελεσμάτων για κάθε χρήστη με βάση τις προτεινόμενες ταινίες και το σύνολο των relevant αντικειμένων του test set. Η συνάρτηση `precision_at_k` υπολογίζει το ποσοστό των προτάσεων στις πρώτες k θέσεις που είναι πράγματι σχετικές. Για τον σκοπό αυτό, ο κώδικας ελέγχει τα πρώτα k στοιχεία της λίστας `recommended_items`, μετρά πόσα από αυτά ανήκουν στο σύνολο `relevant_items` και αποθηκεύει το πλήθος αυτό στη μεταβλητή `hits` και συνεχίζοντας διαιρεί το πλήθος των επιτυχιών με το k , επιστρέφοντας έτσι το Precision@K. Αν για οποιονδήποτε λόγο το k είναι μηδενικό, επιστρέφεται 0.0 ώστε να αποφεύγεται διαίρεση με το μηδέν.

Η συνάρτηση `recall_at_k` μετρά πόσο μεγάλο μέρος των σχετικών ταινιών καταφέρνει να ανακτήσει το σύστημα μέσα στις πρώτες k προτάσεις. Αρχικά ελέγχεται αν το σύνολο `relevant_items` είναι κενό· σε αυτή την περίπτωση επιστρέφεται 0.0, καθώς δεν υπάρχουν σχετικά αντικείμενα προς ανάκτηση. Διαφορετικά, ο κώδικας μετρά ξανά τις επιτυχίες στις πρώτες k προτάσεις και στη συνέχεια διαιρεί αυτό το πλήθος με το συνολικό

πλήθος των relevant αντικειμένων. Έτσι προκύπτει το Recall@K , δηλαδή το ποσοστό κάλυψης των σχετικών ταινιών από τη λίστα προτάσεων.

Η συνάρτηση $\text{average_precision_at_k}$ υπολογίζει τη μέση ακρίβεια της λίστας προτάσεων λαμβάνοντας υπόψη και τη σειρά εμφάνισης των relevant αντικειμένων. Για τον σκοπό αυτό, ο κώδικας διατρέχει διαδοχικά τις πρώτες k προτάσεις με τη βοήθεια της enumerate , κρατώντας τόσο τη θέση i όσο και το αντίστοιχο αντικείμενο. Κάθε φορά που εντοπίζεται relevant ταινία, αυξάνεται ο μετρητής hits και προστίθεται στο precision_sum η ακρίβεια που έχει επιτευχθεί μέχρι εκείνο το σημείο, δηλαδή ο λόγος hits / i . Με αυτόν τον τρόπο, οι relevant ταινίες που εμφανίζονται νωρίτερα στη λίστα συνεισφέρουν περισσότερο στη συνολική τιμή της μετρικής. Στο τέλος, το άθροισμα αυτό διαιρείται με το μικρότερο μεταξύ του πλήθους των relevant αντικειμένων και του k , ώστε να παραχθεί η τελική τιμή του $\text{Average Precision@K}$.

Η συνάρτηση ndcg_at_k υπολογίζει τη μετρική NDCG@K , η οποία αξιολογεί κατά πόσο οι σχετικές ταινίες τοποθετούνται ψηλά στην τελική κατάταξη. Ο κώδικας δημιουργεί αρχικά ένα δυαδικό διάνυσμα y_true , στο οποίο κάθε θέση παίρνει τιμή 1 αν το αντίστοιχο προτεινόμενο αντικείμενο ανήκει στα relevant_items και 0 διαφορετικά. Παράλληλα, δημιουργείται το διάνυσμα y_score , το οποίο περιέχει τα scores των πρώτων k προτάσεων. Τα δύο αυτά διανύσματα μετατρέπονται σε μορφή κατάλληλη για τη συνάρτηση ndcg_score , η οποία υπολογίζει την ποιότητα της κατάταξης δίνοντας μεγαλύτερη βαρύτητα στις πρώτες θέσεις της λίστας.

Στο τέλος, η συνάρτηση hit_rate_at_k υπολογίζει μια απλούστερη αλλά ιδιαίτερα χρήσιμη μετρική, τη HitRate@K . Εδώ ο κώδικας ελέγχει αν υπάρχει έστω ένα relevant αντικείμενο μέσα στις πρώτες k προτάσεις. Αν βρεθεί τουλάχιστον μία επιτυχία, επιστρέφεται 1.0, διαφορετικά 0.0. Η μετρική αυτή αποτυπώνει το αν το σύστημα κατάφερε να προσφέρει τουλάχιστον μία χρήσιμη πρόταση στον χρήστη, ανεξάρτητα από το πλήθος ή την ακριβή θέση των υπόλοιπων relevant αντικειμένων.

```

94 def intra_list_diversity(recommended_items): 1usage new *
95     """
96     Μετρά πόσο διαφορετικές είναι μεταξύ τους οι προτεινόμενες ταινίες.
97     Υπολογίζεται ως:
98     1 - μέσο cosine similarity των TF-IDF vectors.
99     Όσο μεγαλύτερη η τιμή, τόσο μεγαλύτερη η diversity.
100     """
101     if len(recommended_items) < 2:
102         return 0.0
103
104     item_indices = [movie_id_to_idx[item] for item in recommended_items if item in movie_id_to_idx]
105
106     if len(item_indices) < 2:
107         return 0.0
108
109     item_vectors = tfidf_matrix[item_indices]
110     sim_matrix = cosine_similarity(item_vectors)
111
112     n = len(item_indices)
113     pairwise_sims = []
114
115     for i in range(n):
116         for j in range(i + 1, n):
117             pairwise_sims.append(sim_matrix[i, j])
118
119     if not pairwise_sims:
120         return 0.0
121
122     return 1.0 - np.mean(pairwise_sims)

```

Εικόνα 33: Υπολογισμός ποικιλομορφίας στις προτεινόμενες ταινίες

Η συνάρτηση `intra_list_diversity` υπολογίζει τον βαθμό ποικιλίας των ταινιών που περιλαμβάνονται στην τελική λίστα προτάσεων. Ενώ οι κλασικές μετρικές αξιολόγησης, όπως το Precision ή το Recall, εξετάζουν αν οι προτάσεις είναι σχετικές με τις προτιμήσεις του χρήστη, η συγκεκριμένη μετρική εστιάζει στο κατά πόσο οι ίδιες οι προτεινόμενες ταινίες διαφέρουν μεταξύ τους. Αρχικά, ο κώδικας ελέγχει αν η λίστα `recommended_items` περιέχει τουλάχιστον δύο ταινίες, καθώς διαφορετικά δεν είναι δυνατό να μετρηθεί ποικιλία. Έπειτα, κάθε ταινία αντιστοιχίζεται στη θέση της μέσα στη μήτρα `tfidf_matrix` μέσω του λεξικού `movie_id_to_idx`, ώστε να ανακτηθούν τα διανύσματα περιεχομένου των προτεινόμενων ταινιών. Αν μετά από αυτή τη διαδικασία παραμένουν λιγότερες από δύο έγκυρες ταινίες, η συνάρτηση επιστρέφει ξανά 0.0. Ακόμη, εξάγονται από τη `tfidf_matrix` τα αντίστοιχα διανύσματα και υπολογίζεται ο πίνακας ομοιότητας `sim_matrix` με χρήση της `cosine_similarity`, ο οποίος αποτυπώνει πόσο όμοια είναι μεταξύ τους τα προτεινόμενα αντικείμενα. Στο επόμενο βήμα, ο κώδικας υπολογίζει την ομοιότητα ανάμεσα σε όλες τις διαφορετικές δυάδες ταινιών που υπάρχουν στη λίστα προτάσεων. Υπό αυτό το πρίσμα,

χρησιμοποιεί τη μεταβλητή n για να γνωρίζει πόσες ταινίες συμμετέχουν τελικά στον υπολογισμό και δημιουργεί τη λίστα `pairwise_sims`, μέσα στην οποία θα αποθηκευτούν οι επιμέρους τιμές ομοιότητας. Στη συνέχεια, με τη βοήθεια δύο εμφωλευμένων επαναλήψεων, εξετάζονται όλα τα μοναδικά ζεύγη ταινιών της λίστας. Η εσωτερική επανάληψη ξεκινά από τη θέση $i + 1$, γεγονός που εξασφαλίζει δύο πράγματα: πρώτον, ότι δεν υπολογίζεται ποτέ η ομοιότητα μιας ταινίας με τον εαυτό της, και δεύτερον, ότι το ίδιο ζεύγος δεν εξετάζεται δύο φορές με αντίστροφη σειρά. Έτσι, αν για παράδειγμα έχει ήδη ληφθεί υπόψη το ζεύγος μιας ταινίας A με μια ταινία B , δεν χρειάζεται να υπολογιστεί ξανά και το ζεύγος B με A , αφού η τιμή της ομοιότητας είναι η ίδια. Κάθε τέτοια τιμή cosine similarity αποθηκεύεται στη λίστα `pairwise_sims`, η οποία τελικά συγκεντρώνει όλες τις απαραίτητες pairwise ομοιότητες μεταξύ των προτάσεων. Αν για οποιονδήποτε λόγο η λίστα `pairwise_sims` παραμείνει κενή, ο κώδικας επιστρέφει 0.0, ώστε να αποφευχθεί προσπάθεια υπολογισμού μέσου όρου σε άδεια λίστα. Στο τελευταίο βήμα, η τιμή της `diversity` προκύπτει ως 1.0 - μέσος όρος pairwise similarity. Η λογική αυτού του υπολογισμού είναι ότι όσο πιο όμοιες είναι μεταξύ τους οι προτεινόμενες ταινίες, τόσο μεγαλύτερος θα είναι ο μέσος όρος των pairwise similarities και τόσο μικρότερη θα είναι η τελική τιμή της `diversity`. Αντίθετα, όταν οι ταινίες της λίστας διαφέρουν περισσότερο ως προς το περιεχόμενό τους, η μέση ομοιότητα μειώνεται και η `diversity` αυξάνεται. Συνεπώς, η συνάρτηση μετατρέπει τη συνολική εσωτερική ομοιότητα της λίστας σε μέτρο ποικιλίας, αποτυπώνοντας το κατά πόσο οι προτάσεις είναι μεταξύ τους ετερογενείς.

```

127 movies = pd.read_csv(movies_path)
128 tags = pd.read_csv(tags_path)
129 train = pd.read_csv(train_path)
130 test = pd.read_csv(test_path)
131 tmdb = pd.read_csv(tmdb_path)
132
133 # Καθαρίζουμε μόνο τα DataFrames στη μνήμη
134 for col in ["title", "genres"]:
135     if col in train.columns:
136         train = train.drop(columns=[col])
137     if col in test.columns:
138         test = test.drop(columns=[col])
139
140 print("Movies raw:", movies.shape)
141 print("Tags raw:", tags.shape)
142 print("Train raw:", train.shape)
143 print("Test raw:", test.shape)
144 print("TMdb raw:", tmdb.shape)
145
146 # =====
147 # CANDIDATE CATALOG = TRAIN ITEMS ONLY
148 # =====
149 train_item_ids = set(train["movieId"].unique())
150 test_item_ids = set(test["movieId"].unique())
151
152 print("Unique movies in original movies.csv:", movies["movieId"].nunique())
153 print("Unique movies in TRAIN:", len(train_item_ids))
154 print("Unique movies in TEST:", len(test_item_ids))
155
156 movies = movies[movies["movieId"].isin(train_item_ids)].copy()
157
158 print("Candidate movies for recommendation:", movies["movieId"].nunique())

```

Εικόνα 34: Δημιουργία καταλόγου υποψήφιων ταινιών

Ακολούθως, πραγματοποιείται η φόρτωση των βασικών συνόλων δεδομένων που θα χρησιμοποιηθούν στην υλοποίηση του recommender. Συγκεκριμένα, τα αρχεία movies.csv, tags.csv, train.csv, test.csv και tmdb_enriched_movies.csv διαβάζονται σε μορφή DataFrame μέσω της pd.read_csv, ώστε να είναι διαθέσιμα για επεξεργασία στα επόμενα βήματα. Αμέσως μετά, εφαρμόζεται ένας προληπτικός καθαρισμός στα train και test DataFrames, όπου αφαιρούνται οι στήλες title και genres εφόσον υπάρχουν. Η ενέργεια αυτή δεν τροποποιεί τα αρχικά αρχεία, αλλά μόνο τα δεδομένα στη μνήμη, και αποσκοπεί στην αποφυγή πλεονασμού ή πιθανών συγκρούσεων με τα αντίστοιχα πεδία του κύριου πίνακα movies, ο οποίος αποτελεί

τη βασική πηγή μεταδεδομένων των ταινιών. Στη συνέχεια, ο κώδικας εκτυπώνει τις διαστάσεις των πέντε DataFrames μέσω της ιδιότητας `shape`, ώστε να επιβεβαιωθεί ότι τα δεδομένα φορτώθηκαν σωστά και να υπάρχει σαφής εικόνα του αρχικού μεγέθους κάθε πηγής πληροφορίας.

Αμέσως μετά έρχεται ο ορισμός του `candidate catalog`, δηλαδή του συνόλου των ταινιών από το οποίο το σύστημα θα μπορεί τελικά να παράγει προτάσεις. Ως εκ τούτου, εξάγονται πρώτα τα μοναδικά `movieId` που εμφανίζονται στο `training set` και στο `test set`, και αποθηκεύονται στα σύνολα `train_item_ids` και `test_item_ids`. Κατόπιν, ο κώδικας εκτυπώνει το πλήθος των μοναδικών ταινιών στο αρχικό `movies.csv`, καθώς και το πλήθος των μοναδικών ταινιών που εμφανίζονται στο `train` και στο `test`, ώστε να καταστεί εμφανές πόσο διαφέρει το πλήρες σύνολο ταινιών από εκείνο που συμμετέχει πραγματικά στο πειραματικό σχήμα. Έστερα, ο πίνακας `movies` φιλτράρεται έτσι ώστε να διατηρηθούν μόνο οι ταινίες των οποίων το `movieId` εμφανίζεται στο `training set`. Έτσι, ο τελικός κατάλογος υποψήφιων προτάσεων περιορίζεται αποκλειστικά σε αντικείμενα που ανήκουν στο `train split`. Η απόφαση αυτή είναι κρίσιμη, καθώς εξασφαλίζει ότι το μοντέλο παράγει συστάσεις μόνο για ταινίες για τις οποίες διαθέτει πληροφορία κατά το στάδιο της εκπαίδευσης και της κατασκευής των χαρακτηριστικών. Ολοκληρώνοντας, εκτυπώνεται το πλήθος των μοναδικών ταινιών που παραμένουν μετά το φιλτράρισμα, δίνοντας έτσι το τελικό μέγεθος του `candidate catalog` από το οποίο θα αντλούνται οι προτάσεις.

```

163 movies["genres"] = movies["genres"].fillna("").replace("no genres listed", "")
164 movies["genres_text"] = movies["genres"].str.replace("|", " ", regex=False).apply(clean_text)
165
166 movies["title"] = movies["title"].fillna("")
167 movies["title_text"] = movies["title"].apply(clean_text)
168
169 # =====
170 # CLEAN TAGS
171 # =====
172 tags["tag"] = tags["tag"].fillna("").astype(str).apply(clean_text)
173 tags = tags[tags["movieId"].isin(train_item_ids)].copy()
174 tags = tags[tags["tag"].str.len() >= 3]
175 tags = tags[~tags["tag"].str.fullmatch(r"\d+")]
176
177 tag_counts = tags["tag"].value_counts()
178 valid_tags = tag_counts[tag_counts >= 10].index
179 tags = tags[tags["tag"].isin(valid_tags)].copy()
180
181 print("Tags after cleaning:", tags.shape)
182
183 movie_tags = tags.groupby("movieId")["tag"].apply(lambda x: " ".join(x)).reset_index()
184 movie_tags.rename(columns={"tag": "tags_text"}, inplace=True)
185
186 movies = movies.merge(movie_tags, on="movieId", how="left")
187 movies["tags_text"] = movies["tags_text"].fillna("")

```

Εικόνα 35: Καθαρισμός ειδών και ετικετών ταινιών

Κατά την επόμενη φάση της διαδικασίας πραγματοποιείται ο καθαρισμός και η προετοιμασία των βασικών λεκτικών χαρακτηριστικών των ταινιών. Πρωτίστως, στη στήλη genres του πίνακα movies, οι κενές τιμές αντικαθίστανται με κενό string, ενώ η ειδική τιμή "(no genres listed)" αφαιρείται ώστε να μη θεωρείται πραγματική πληροφορία περιεχομένου. Στη συνέχεια δημιουργείται η νέα στήλη genres_text, όπου οι διαχωριστές «|» αντικαθίστανται με κενά και το αποτέλεσμα περνά από τη συνάρτηση clean_text. Έτσι, τα genres μετατρέπονται σε καθαρό και ομοιόμορφο κείμενο, κατάλληλο για τη μετέπειτα διανυσματική αναπαράσταση. Αντίστοιχα, στη στήλη title οι κενές τιμές αντικαθίστανται επίσης με κενό string και δημιουργείται η νέα στήλη title_text, η οποία περιέχει τον καθαρισμένο τίτλο κάθε ταινίας. Με αυτόν τον τρόπο, τόσο τα genres όσο και ο τίτλος μετατρέπονται σε συνεπή λεκτικά χαρακτηριστικά που μπορούν να αξιοποιηθούν ενιαία στο μοντέλο.

Στη συνέχεια ακολουθεί ο καθαρισμός των tags, τα οποία αποτελούν μια επιπλέον πηγή περιγραφικής πληροφορίας για τις ταινίες. Αρχικά, η στήλη tag καθαρίζεται με παρόμοιο τρόπο: οι κενές τιμές αντικαθίστανται, το πεδίο μετατρέπεται σε συμβολοσειρά και εφαρμόζεται η clean_text, ώστε όλα τα tags να βρίσκονται σε ενιαία μορφή. Έπειτα,

διατηρούνται μόνο τα tags που αντιστοιχούν σε ταινίες του training catalog, ώστε να υπάρχει πλήρης συνέπεια με το σύνολο των αντικειμένων που μπορούν να προταθούν. Ακολουθεί πρόσθετο φιλτράρισμα, όπου απορρίπτονται tags με μήκος μικρότερο από τρεις χαρακτήρες, καθώς και tags που αποτελούνται αποκλειστικά από αριθμούς, επειδή τέτοιες τιμές συνήθως έχουν πολύ μικρή περιγραφική αξία. Στη συνέχεια υπολογίζεται η συχνότητα εμφάνισης κάθε tag και κρατούνται μόνο όσα εμφανίζονται τουλάχιστον δέκα φορές στο σύνολο των δεδομένων. Με αυτή την προσέγγιση αφαιρούνται πολύ σπάνια ή θορυβώδη tags, τα οποία θα μπορούσαν να επιβαρύνουν τη διανυσματική αναπαράσταση χωρίς να προσφέρουν ουσιαστική πληροφορία. Η εκτύπωση του tags.shape μετά το φιλτράρισμα επιτρέπει την επιβεβαίωση του τελικού μεγέθους του πίνακα tags μετά τον καθαρισμό.

Συνοψίζοντας, τα καθαρισμένα tags ομαδοποιούνται ανά movieId, έτσι ώστε όλα τα tags της ίδιας ταινίας να συνενωθούν σε ένα ενιαίο κειμενικό πεδίο. Το αποτέλεσμα αποθηκεύεται στη νέα στήλη tags_text, η οποία στη συνέχεια συγχωνεύεται με τον πίνακα movies μέσω του movieId. Μετά το merge, όσες ταινίες δεν διαθέτουν tags λαμβάνουν κενό string, ώστε να μην προκύπτουν κενές τιμές στα επόμενα στάδια. Έτσι, κάθε ταινία εμπλουτίζεται με ένα επιπλέον λεκτικό χαρακτηριστικό που συνοψίζει τη συλλογική πληροφορία των tags και ενισχύει την περιγραφική της αναπαράσταση.

```

192     tmdb_cols = [
193         "movieId",
194         "overview",
195         "tagline",
196         "keywords_tmdb",
197         "director",
198         "cast",
199         "production_companies",
200         "collection",
201         "original_title",
202         "genres_tmdb"
203     ]
204
205     tmdb = tmdb[tmdb_cols].copy()
206     tmdb = tmdb[tmdb["movieId"].isin(train_item_ids)].copy()
207
208     for col in tmdb_cols[1:]:
209         tmdb[col] = tmdb[col].fillna("").apply(clean_text)
210
211     movies = movies.merge(tmdb, on="movieId", how="left")

```

Εικόνα 36: Ενσωμάτωση μεταδεδομένων TMDb

Στο επόμενο στάδιο ενσωματώνονται στον βασικό πίνακα `movies` τα εμπλουτισμένα μεταδεδομένα που προέρχονται από το TMDb. Αρχικά, στη λίστα `tmdb_cols` ορίζονται τα πεδία που θα χρησιμοποιηθούν από το αρχείο εμπλουτισμού, δηλαδή το `movieId` και μια σειρά από περιγραφικά χαρακτηριστικά όπως `overview`, `tagline`, `keywords_tmdb`, `director`, `cast`, `production_companies`, `collection`, `original_title` και `genres_tmdb`. Τα πεδία αυτά επιλέγονται επειδή παρέχουν πολύ πλουσιότερη πληροφορία για το περιεχόμενο και την ταυτότητα κάθε ταινίας σε σχέση με τα βασικά δεδομένα του MovieLens.

Κατόπιν, ο πίνακας `tmdb` περιορίζεται μόνο στις συγκεκριμένες στήλες και έπειτα φιλτράρεται ώστε να διατηρηθούν μόνο οι ταινίες των οποίων το `movieId` ανήκει στο `train_item_ids`. Μέσω αυτής της διαδικασίας, ο εμπλουτισμός παραμένει απολύτως ευθυγραμμισμένος με το candidate catalog του συστήματος και δεν εισάγονται πληροφορίες για ταινίες που δεν συμμετέχουν τελικά στο σύνολο υποψήφιων προτάσεων. Ακολουθεί

καθαρισμός όλων των κειμενικών πεδίων του TMDb, εκτός από το `movieId`, με αντικατάσταση των κενών τιμών από κενό string και εφαρμογή της συνάρτησης `clean_text`, ώστε όλα τα μεταδεδομένα να αποκτήσουν ενιαία και καθαρή μορφή πριν χρησιμοποιηθούν στη σύνθεση του περιεχομένου κάθε ταινίας.

Εν κατακλείδι, τα εμπλουτισμένα αυτά χαρακτηριστικά συγχωνεύονται με τον πίνακα `movies` μέσω του `movieId` με χρήση αριστερής σύνδεσης (`how="left"`), ώστε να διατηρηθούν όλες οι ταινίες του `candidate catalog` και να προστεθούν σε καθεμία, όπου υπάρχουν διαθέσιμα, τα αντίστοιχα πεδία από το TMDb. Με αυτό το βήμα, ο βασικός πίνακας ταινιών παύει να στηρίζεται μόνο σε πληροφορίες όπως ο τίτλος, τα `genres` και τα `tags`, και αποκτά ένα πολύ πιο πλούσιο περιγραφικό υπόβαθρο, το οποίο θα αξιοποιηθεί αργότερα για τη δημιουργία της σύνθετης αναπαράστασης `rich_content`.

```
213     for col in [  
214         "overview",  
215         "tagline",  
216         "keywords_tmdb",  
217         "director",  
218         "cast",  
219         "production_companies",  
220         "collection",  
221         "original_title",  
222         "genres_tmdb"  
223     ]:  
224         movies[col] = movies[col].fillna("")  
225  
226     movies["overview"] = movies["overview"].apply(lambda x: truncate_words(x, max_words: 60))
```

Εικόνα 37: Διαχείριση ελλিপών τιμών στα μεταδεδομένα TMDb

Στο αμέσως επόμενο βήμα γίνεται πρόσθετη επεξεργασία των πεδίων που προήλθαν από το TMDb μετά τη συγχώνευσή τους με τον πίνακα `movies`. Συγκεκριμένα, για τις στήλες `overview`, `tagline`, `keywords_tmdb`, `director`, `cast`, `production_companies`, `collection`, `original_title` και `genres_tmdb`, οι τυχόν κενές τιμές αντικαθίστανται με κενό string. Η ενέργεια αυτή είναι απαραίτητη, επειδή μετά το `merge` δεν είναι βέβαιο ότι όλες οι ταινίες διαθέτουν πληροφορία σε κάθε πεδίο του TMDb. Με την αντικατάσταση των ελλειπουσών τιμών εξασφαλίζεται ότι όλα τα πεδία θα μπορούν να χρησιμοποιηθούν χωρίς σφάλματα στα επόμενα στάδια σύνθεσης του κειμενικού περιεχομένου.

Αμέσως μετά, εφαρμόζεται ειδική επεξεργασία στη στήλη `overview`, η οποία συνήθως περιέχει πολύ εκτενείς περιγραφές. Με τη χρήση της `truncate_words(x, 60)`, κάθε

περιγραφή περιορίζεται στις πρώτες 60 λέξεις. Ο περιορισμός αυτός βοηθά ώστε το overview να παραμένει χρήσιμο ως πηγή πληροφορίας, χωρίς όμως να κυριαρχεί δυσανάλογα στη συνολική αναπαράσταση της ταινίας. Έτσι, διατηρείται μια πιο ισορροπημένη συμβολή των διαφορετικών χαρακτηριστικών στο τελικό περιεχόμενο που θα χρησιμοποιηθεί από το μοντέλο.

```
231 movies["rich_content"] = (
232     (movies["title_text"] + " ") * 2 +
233     (movies["genres_text"] + " ") * 3 +
234     (movies["genres_tmdb"] + " ") * 3 +
235     (movies["keywords_tmdb"] + " ") * 2 +
236     (movies["director"] + " ") * 2 +
237     (movies["cast"] + " ") * 2 +
238     (movies["collection"] + " ") * 2 +
239     (movies["tags_text"] + " ") * 1 +
240     (movies["tagline"] + " ") * 1 +
241     (movies["overview"] + " ") * 1 +
242     (movies["original_title"] + " ") * 1
243 ).str.strip()
244
245 movies["rich_content"] = movies["rich_content"].replace("", "unknown")
```

Εικόνα 38: Δημιουργία εμπλουτισμένου κειμενικού περιεχομένου για κάθε ταινία

Στο σημείο αυτό δημιουργείται η στήλη rich_content, η οποία αποτελεί τη σύνθετη περιγραφική αναπαράσταση κάθε ταινίας και συγκεντρώνει σε ενιαία μορφή όλα τα λεκτικά χαρακτηριστικά που έχουν προετοιμαστεί στα προηγούμενα βήματα. Ο κώδικας ενώνει τα πεδία title_text, genres_text, genres_tmdb, keywords_tmdb, director, cast, collection, tags_text, tagline, overview και original_title σε ένα ενιαίο κείμενο, το οποίο θα χρησιμοποιηθεί αργότερα για τη δημιουργία των TF-IDF διανυσμάτων. Ωστόσο, η σύνθεση αυτή δεν γίνεται με ίση συμμετοχή όλων των πεδίων. Ορισμένα χαρακτηριστικά επαναλαμβάνονται περισσότερες φορές μέσα στο τελικό string, ώστε να αποκτήσουν μεγαλύτερη βαρύτητα στη διανυσματική αναπαράσταση. Ουσιαστικότερα, ο τίτλος διπλασιάζεται, τα genres από το MovieLens και το TMDb τριπλασιάζονται, ενώ τα keywords_tmdb, director, cast και collection διπλασιάζονται επίσης. Αντίθετα, τα tags_text, tagline, overview και original_title συμμετέχουν μία φορά. Με αυτή τη λογική, ο κώδικας πραγματοποιεί μια μορφή χειροκίνητης στάθμισης των χαρακτηριστικών πριν από την

εφαρμογή του TF-IDF, ενισχύοντας πεδία που θεωρούνται πιο αντιπροσωπευτικά για το περιεχόμενο και την ταυτότητα της ταινίας.

Η προσέγγιση αυτή είναι ιδιαίτερα σημαντική, επειδή επιτρέπει στο μοντέλο να μην αντιμετωπίζει όλα τα λεκτικά πεδία ως ισοδύναμα. Για παράδειγμα, τα genres και οι λέξεις-κλειδιά έχουν συνήθως πιο άμεση σχέση με το θεματικό περιεχόμενο μιας ταινίας από ό,τι ένα tagline ή ένας εναλλακτικός τίτλος, γι' αυτό και ενισχύονται περισσότερο μέσα στο rich_content. Μετά τη συνένωση όλων των πεδίων, εφαρμόζεται η μέθοδος .str.strip() ώστε να αφαιρεθούν τυχόν περιττά κενά από την αρχή και το τέλος του τελικού κειμένου. Τέλος, σε περιπτώσεις όπου το rich_content καταλήγει κενό, αντικαθίσταται από τη λέξη "unknown". Η τελευταία αυτή ενέργεια λειτουργεί προληπτικά, διασφαλίζοντας ότι κάθε ταινία θα διαθέτει κάποια μορφή κειμενικής αναπαράστασης και ότι δεν θα προκύψουν προβλήματα στο στάδιο της διανυσματοποίησης.

```

250     tfidf = TfidfVectorizer(
251         stop_words="english",
252         max_features=80000,
253         ngram_range=(1, 2),
254         min_df=7,
255         max_df=0.8,
256         sublinear_tf=True
257     )
258
259     tfidf_matrix = tfidf.fit_transform(movies["rich_content"])
260     print("TF-IDF matrix:", tfidf_matrix.shape)
261
262     movie_id_to_idx = pd.Series(movies.index, index=movies["movieId"]).to_dict()
263
264     train = train[train["movieId"].isin(movie_id_to_idx)].copy()
265     test = test[test["movieId"].isin(movie_id_to_idx)].copy()
266
267     print("Train after alignment:", train.shape)
268     print("Test after alignment:", test.shape)
269     print("Final candidate movie count:", len(movie_id_to_idx))
270
271     # =====
272     # POPULARITY FROM TRAIN ONLY
273     # =====
274     movie_popularity = train["movieId"].value_counts()
275     movie_popularity = np.log1p(movie_popularity)
276     movie_popularity = movie_popularity / movie_popularity.max()
277     movie_popularity = movie_popularity.to_dict()

```

Εικόνα 39: Δημιουργία TF-IDF πίνακα και υπολογισμός δημοφιλίας ταινιών

Στο στάδιο που ακολουθεί ορίζεται και εφαρμόζεται ο `TfidfVectorizer`, ο οποίος μετατρέπει το πεδίο `rich_content` κάθε ταινίας σε αριθμητική διανυσματική αναπαράσταση. Η παραμετροποίησή του έχει γίνει έτσι ώστε να διατηρείται ισορροπία ανάμεσα στον πλούτο της πληροφορίας και στον περιορισμό του θορύβου. Συγκεκριμένα, με την επιλογή `stop_words="english"` αφαιρούνται πολύ συχνές αγγλικές λέξεις χωρίς ιδιαίτερη διακριτική αξία, ενώ το `max_features=80000` περιορίζει το συνολικό λεξιλόγιο στα 80.000 σημαντικότερα χαρακτηριστικά. Η παράμετρος `ngram_range=(1, 2)` επιτρέπει τη χρήση τόσο μεμονωμένων όρων όσο και διλέκτων φράσεων, ώστε το μοντέλο να αποτυπώνει όχι μόνο λέξεις αλλά και συχνούς συνδυασμούς τους. Παράλληλα, με το `min_df=7` απορρίπτονται πολύ σπάνιοι όροι που εμφανίζονται σε λιγότερες από επτά ταινίες, ενώ με το `max_df=0.8` αφαιρούνται όροι που

εμφανίζονται σε υπερβολικά μεγάλο ποσοστό του corpus και άρα δεν συμβάλλουν ουσιαστικά στη διάκριση των αντικειμένων. Τέλος, η επιλογή `sublinear_tf=True` εφαρμόζει λογαριθμική κλιμάκωση στη συχνότητα εμφάνισης των όρων, ώστε η πολύ υψηλή επανάληψη ενός όρου μέσα στο ίδιο κείμενο να μην ενισχύει δυσανάλογα το τελικό βάρος του.

Αφού οριστεί ο `vectorizer`, εφαρμόζεται η μέθοδος `fit_transform` στη στήλη `movies["rich_content"]`, παράγοντας τη μήτρα `tfidf_matrix`. Η μήτρα αυτή αποτελεί τη βασική αναπαράσταση περιεχομένου των ταινιών, όπου κάθε γραμμή αντιστοιχεί σε μία ταινία και κάθε στήλη σε ένα χαρακτηριστικό του λεξιλογίου που έχει εξαχθεί από το corpus. Η εκτύπωση του σχήματος της μήτρας επιτρέπει την επιβεβαίωση του τελικού πλήθους ταινιών και χαρακτηριστικών που προέκυψαν μετά τη διανυσματοποίηση. Στη συνέχεια, δημιουργείται το λεξικό `movie_id_to_idx`, το οποίο αντιστοιχίζει κάθε `movieId` στη γραμμή που καταλαμβάνει μέσα στη `tfidf_matrix`. Το λεξικό αυτό είναι κρίσιμο για τα επόμενα στάδια, επειδή επιτρέπει τη γρήγορη μετάβαση από το αναγνωριστικό μιας ταινίας στη διανυσματική της αναπαράσταση.

Έπειτα, τα `DataFrames` `train` και `test` ευθυγραμμίζονται με τον τελικό κατάλογο ταινιών που υπάρχουν στη `tfidf_matrix`, διατηρώντας μόνο τις εγγραφές των οποίων το `movieId` περιλαμβάνεται στο `movie_id_to_idx`. Με αυτόν τον τρόπο εξασφαλίζεται ότι όλες οι αξιολογήσεις που παραμένουν στα δύο σύνολα αντιστοιχούν σε ταινίες για τις οποίες υπάρχει διαθέσιμη περιγραφική αναπαράσταση. Οι εκτυπώσεις που ακολουθούν εμφανίζουν το μέγεθος των `train` και `test` μετά την ευθυγράμμιση, καθώς και το τελικό πλήθος των υποψήφιων ταινιών, παρέχοντας σαφή εικόνα για τη βάση πάνω στην οποία θα λειτουργήσει το recommender.

Στην τελευταία φάση, υπολογίζεται μια απλή μετρική δημοφιλίας αποκλειστικά από το training set. Ως αφετηρία, μετρώνται οι εμφανίσεις κάθε `movieId` στο `train`, δηλαδή πόσες αξιολογήσεις έχει δεχθεί κάθε ταινία. Στη συνέχεια, η πληροφορία αυτή μετασχηματίζεται με τη συνάρτηση `np.log1p`, ώστε να περιοριστεί η υπερβολική επίδραση πολύ δημοφιλών τίτλων. Ακολουθεί κανονικοποίηση ως προς τη μέγιστη τιμή, ώστε όλες οι τιμές δημοφιλίας να τοποθετηθούν σε κοινή κλίμακα, και τέλος το αποτέλεσμα μετατρέπεται σε λεξικό (`movie_popularity`) για γρήγορη ανάκτηση. Η πληροφορία αυτή θα χρησιμοποιηθεί αργότερα ως συμπληρωματικός παράγοντας στο τελικό recommendation score, επιτρέποντας στο σύστημα να συνδυάζει την ομοιότητα περιεχομένου με έναν ελεγχόμενο βαθμό δημοφιλίας.

```

282 user_train_dict = {uid: df.copy() for uid, df in train.groupby("userId")}
283
284 def build_user_profile(user_id): 1 usage new *
285     user_train = user_train_dict.get(user_id)
286
287     if user_train is None or user_train.empty:
288         return None, None
289
290     user_likes = user_train[user_train["rating"] >= PROFILE_THRESHOLD].copy()
291     if user_likes.empty:
292         return None, None
293
294     user_likes = user_likes[user_likes["movieId"].isin(movie_id_to_idx)].copy()
295     if user_likes.empty:
296         return None, None
297
298     item_indices = user_likes["movieId"].map(movie_id_to_idx).to_numpy()
299     ratings_array = user_likes["rating"].to_numpy()
300
301     weights = get_rating_weights(ratings_array, mode=WEIGHT_MODE)
302
303     item_matrix = tfidf_matrix[item_indices]
304     weighted_matrix = item_matrix.multiply(weights)
305
306     weight_sum = weights.sum()
307     if weight_sum == 0:
308         return None, None
309
310     user_profile = np.asarray(weighted_matrix.sum(axis=0) / weight_sum).reshape(1, -1)
311     seen_movie_ids = set(user_train["movieId"].unique())
312
313     return user_profile, seen_movie_ids

```

Εικόνα 40: Δημιουργία προφίλ χρήστη

Στο σημείο αυτό προετοιμάζεται και κατασκευάζεται το προφίλ προτιμήσεων κάθε χρήστη, το οποίο αποτελεί τον πυρήνα της content-based λογικής του συστήματος. Αρχικά δημιουργείται το λεξικό `user_train_dict`, στο οποίο κάθε `userId` αντιστοιχίζεται στο υποσύνολο του `train` που περιέχει μόνο τις δικές του αξιολογήσεις. Η επιλογή αυτή βελτιώνει την αποδοτικότητα του κώδικα, καθώς επιτρέπει άμεση ανάκτηση του ιστορικού ενός χρήστη χωρίς να απαιτείται νέο φιλτράρισμα του πλήρους `DataFrame` σε κάθε κλήση.

Η συνάρτηση `build_user_profile(user_id)` αναλαμβάνει να δημιουργήσει το διανυσματικό προφίλ ενός συγκεκριμένου χρήστη. Αρχικά ανακτάται από το `user_train_dict` το σύνολο των αξιολογήσεών του. Αν δεν υπάρχει ιστορικό ή αν αυτό είναι κενό, η συνάρτηση επιστρέφει `None, None`, δηλώνοντας ότι δεν είναι δυνατή η κατασκευή προφίλ. Στη συνέχεια,

από το ιστορικό αυτό διατηρούνται μόνο οι ταινίες με βαθμολογία ίση ή μεγαλύτερη από το `PROFILE_THRESHOLD`, δηλαδή μόνο οι αξιολογήσεις που θεωρούνται επαρκώς θετικές ώστε να εκφράζουν πραγματική προτίμηση του χρήστη. Αν και μετά από αυτό το φιλτράρισμα δεν απομένουν ταινίες, η συνάρτηση τερματίζει ξανά επιστρέφοντας κενό αποτέλεσμα. Επιπλέον, διατηρούνται μόνο οι θετικά αξιολογημένες ταινίες που υπάρχουν στο `movie_id_to_idx`, δηλαδή μόνο όσες διαθέτουν έγκυρη αντιστοίχιση στη `tfidf_matrix`. Αυτό εξασφαλίζει ότι κάθε ταινία που θα συμμετάσχει στη δημιουργία του προφίλ έχει διαθέσιμη διανυσματική αναπαράσταση.

Στο επόμενο βήμα, τα `movieId` των θετικά αξιολογημένων ταινιών μετατρέπονται στις αντίστοιχες θέσεις τους μέσα στη `tfidf_matrix`, ενώ οι βαθμολογίες τους αποθηκεύονται στον πίνακα `ratings_array`. Πάνω σε αυτές τις βαθμολογίες εφαρμόζεται η συνάρτηση `get_rating_weights` με παράμετρο `mode=WEIGHT_MODE`, ώστε να παραχθούν τα βάρη με τα οποία κάθε ταινία θα επηρεάσει το τελικό προφίλ του χρήστη. Εφόσον στη συγκεκριμένη υλοποίηση έχει επιλεγεί το `WEIGHT_MODE = "discrete_strong"`, οι πολύ υψηλές αξιολογήσεις αποκτούν μεγαλύτερη επιρροή από τις απλώς θετικές, δίνοντας έμφαση στις ταινίες που εκφράζουν πιο έντονα τις προτιμήσεις του χρήστη.

Ακολουθεί η εξαγωγή από τη `tfidf_matrix` των διανυσμάτων που αντιστοιχούν στις επιλεγμένες ταινίες και ο πολλαπλασιασμός τους με τα αντίστοιχα βάρη, μέσω της μεταβλητής `weighted_matrix`. Με αυτόν τον τρόπο, κάθε ταινία συμβάλλει στο τελικό προφίλ ανάλογα όχι μόνο με το περιεχόμενό της αλλά και με το πόσο θετικά την έχει αξιολογήσει ο χρήστης. Στη συνέχεια υπολογίζεται το άθροισμα των βαρών (`weight_sum`) και, εφόσον αυτό είναι μη μηδενικό, κατασκευάζεται το `user_profile` ως ο σταθμισμένος μέσος όρος των TF-IDF διανυσμάτων των θετικά αξιολογημένων ταινιών. Ουσιαστικά, το προφίλ αυτό συνοψίζει τα βασικά χαρακτηριστικά του περιεχομένου που φαίνεται να προτιμά ο χρήστης. Τέλος, δημιουργείται το σύνολο `seen_movie_ids`, το οποίο περιλαμβάνει όλα τα `movieId` που έχει ήδη αξιολογήσει ο χρήστης στο `training set`, ώστε αργότερα οι ταινίες αυτές να αποκλείονται από τη λίστα προτάσεων. Η συνάρτηση επιστρέφει τόσο το διανυσματικό προφίλ του χρήστη όσο και το σύνολο των ήδη προβεβλημένων ταινιών, παρέχοντας έτσι όλα τα απαραίτητα στοιχεία για το στάδιο της παραγωγής συστάσεων.

```

318 def recommend_for_user(user_id, top_k=10, use_popularity=False): 2 usages new *
319     user_profile, seen_movie_ids = build_user_profile(user_id)
320
321     if user_profile is None:
322         return pd.DataFrame(columns=["movieId", "title", "genres", "score"])
323
324     scores = cosine_similarity(user_profile, tfidf_matrix).flatten()
325
326     if use_popularity:
327         pop_scores = np.array(
328             [movie_popularity.get(mid, 0.0) for mid in movies["movieId"]],
329             dtype=np.float32
330         )
331         scores = (1.0 - POPULARITY_WEIGHT) * scores + POPULARITY_WEIGHT * pop_scores
332
333     candidate_mask = ~movies["movieId"].isin(seen_movie_ids)
334     candidate_indices = np.where(candidate_mask)[0]
335     candidate_scores = scores[candidate_indices]
336
337     top_local = np.argsort(candidate_scores)[::-1][:top_k]
338     top_indices = candidate_indices[top_local]
339
340     recs = movies.iloc[top_indices][["movieId", "title", "genres"]].copy()
341     recs["score"] = scores[top_indices]
342
343     return recs.reset_index(drop=True)

```

Εικόνα 41: Παραγωγή συστάσεων ανά χρήστη

Η συνάρτηση `recommend_for_user` υλοποιεί το βασικό στάδιο παραγωγής προτάσεων για έναν συγκεκριμένο χρήστη. Αρχικά, καλείται η `build_user_profile(user_id)`, από την οποία ανακτώνται τόσο το διανυσματικό προφίλ του χρήστη όσο και το σύνολο `seen_movie_ids` με τις ταινίες που έχει ήδη αξιολογήσει στο training set. Αν δεν είναι δυνατό να κατασκευαστεί προφίλ, δηλαδή αν η συνάρτηση επιστρέψει `None`, τότε η `recommend_for_user` επιστρέφει ένα κενό `DataFrame` με τις στήλες `movieId`, `title`, `genres` και `score`, δηλώνοντας ουσιαστικά ότι δεν μπορούν να παραχθούν έγκυρες προτάσεις για τον συγκεκριμένο χρήστη.

Εφόσον υπάρχει διαθέσιμο προφίλ, ο κώδικας υπολογίζει την `cosine similarity` ανάμεσα στο `user_profile` και σε όλα τα διανύσματα της `tfidf_matrix`. Το αποτέλεσμα είναι ένας πίνακας `scores`, στον οποίο κάθε θέση αντιστοιχεί σε μία ταινία του candidate catalog και εκφράζει τον βαθμό ομοιότητας του περιεχομένου της με τις συνολικές προτιμήσεις του χρήστη. Με άλλα λόγια, όσο μεγαλύτερη είναι η τιμή του `score`, τόσο περισσότερο η

συγκεκριμένη ταινία μοιάζει, σε επίπεδο περιεχομένου, με το προφίλ που έχει προκύψει από τις θετικά αξιολογημένες ταινίες του χρήστη.

Αν η παράμετρος `use_popularity` είναι ενεργοποιημένη, τότε στο καθαρά content-based score προστίθεται και η πληροφορία δημοφιλίας. Για τον σκοπό αυτό, δημιουργείται ο πίνακας `pop_scores`, ο οποίος περιέχει για κάθε ταινία του πίνακα `movies` την κανονικοποιημένη τιμή δημοφιλίας της από το λεξικό `monie_popularity`, ή 0.0 αν δεν υπάρχει σχετική καταχώριση. Στη συνέχεια, το τελικό score κάθε ταινίας υπολογίζεται ως γραμμικός συνδυασμός της cosine similarity και της popularity, με βάρη 1.0 - `POPULARITY_WEIGHT` και `POPULARITY_WEIGHT` αντίστοιχα. Στη συγκεκριμένη υλοποίηση, όπου το `POPULARITY_WEIGHT` έχει οριστεί σε 0.05, η ομοιότητα περιεχομένου παραμένει ο κυρίαρχος παράγοντας, ενώ η δημοφιλία λειτουργεί μόνο συμπληρωματικά.

Μετάπειτα προχωράμε στο στάδιο αποκλεισμού των ήδη γνωστών ταινιών. Με χρήση του `candidate_mask` εντοπίζονται οι ταινίες του πίνακα `movies` που δεν ανήκουν στο σύνολο `seen_movie_ids`, δηλαδή όσες δεν έχουν ήδη αξιολογηθεί από τον χρήστη. Έτσι, το σύστημα περιορίζεται μόνο σε νέες υποψήφιες προτάσεις. Οι θέσεις αυτών των ταινιών αποθηκεύονται στο `candidate_indices` και τα αντίστοιχα scores στο `candidate_scores`. Στη συνέχεια, με τη χρήση της `np.argsort(candidate_scores)[::-1][:top_k]` οι υποψήφιες ταινίες ταξινομούνται σε φθίνουσα σειρά σκορ και επιλέγονται οι `top_k` καλύτερες. Οι τελικές θέσεις τους στη μήτρα ταινιών αποθηκεύονται στο `top_indices`.

Στο τελευταίο στάδιο, ο κώδικας δημιουργεί το `DataFrame` `recs`, το οποίο περιλαμβάνει από τον πίνακα `movies` μόνο τις επιλεγμένες ταινίες και μόνο τις στήλες `movieId`, `title` και `genres`. Στη συνέχεια προστίθεται η στήλη `score`, η οποία περιέχει το τελικό σκορ κάθε πρότασης. Τέλος, το `DataFrame` επιστρέφεται με ανανέωση των δεικτών μέσω της `reset_index(drop=True)`, ώστε η λίστα προτάσεων να έχει καθαρή και συνεχόμενη αρίθμηση. Με αυτόν τον τρόπο, η συνάρτηση παράγει για κάθε χρήστη μια τελική κατάταξη νέων ταινιών, ταξινομημένων από την περισσότερο προς τη λιγότερο σχετική σύμφωνα με το περιεχόμενο και, προαιρετικά, με έναν μικρό βαθμό ενίσχυσης από τη δημοφιλία.

```

348 def evaluate_model(sample_size=300, top_k=10, eval_threshold=4.0, use_popularity=False):
349     precision_scores = []
350     recall_scores = []
351     map_scores = []
352     ndcg_scores_list = []
353     hitrate_scores = []
354     diversity_scores = []
355
356     unique_test_users = test["userId"].drop_duplicates()
357     sample_users = unique_test_users.sample(
358         min(sample_size, len(unique_test_users)),
359         random_state=42
360     )
361
362     evaluated_users = 0

```

Εικόνα 42: Αξιολόγηση συστήματος συστάσεων

Η συνάρτηση `evaluate_model` ορίζει το συνολικό πλαίσιο αξιολόγησης του συστήματος προτάσεων. Δέχεται ως παραμέτρους το μέγεθος του δείγματος χρηστών (`sample_size`), το πλήθος των προτάσεων που θα αξιολογηθούν (`top_k`), το κατώφλι με το οποίο μια ταινία θεωρείται *relevant* στο test set (`eval_threshold`) και μια λογική μεταβλητή (`use_popularity`) που καθορίζει αν οι προτάσεις θα βασιστούν αποκλειστικά στην ομοιότητα περιεχομένου ή αν θα ενσωματώνουν και τον παράγοντα δημοφιλίας. Στην αρχή της συνάρτησης δημιουργούνται έξι ξεχωριστές λίστες, `precision_scores`, `recall_scores`, `map_scores`, `ndcg_scores_list`, `hitrate_scores` και `diversity_scores`, οι οποίες θα χρησιμοποιηθούν για την αποθήκευση των επιμέρους τιμών των αντίστοιχων μετρικών για κάθε χρήστη που τελικά θα αξιολογηθεί.

Σε συνέχεια της διαδικασίας, από το `DataFrame` `test` εξάγονται οι μοναδικοί χρήστες μέσω της `drop_duplicates()`, ώστε κάθε χρήστης να εμφανίζεται μόνο μία φορά στη διαδικασία αξιολόγησης. Από αυτό το σύνολο επιλέγεται έπειτα ένα τυχαίο δείγμα χρηστών με χρήση της `sample()`. Το μέγεθος του δείγματος καθορίζεται από το μικρότερο μεταξύ της τιμής `sample_size` και του συνολικού αριθμού των διαθέσιμων χρηστών, ώστε να αποφεύγεται σφάλμα σε περίπτωση που ζητηθεί μεγαλύτερο δείγμα από το πραγματικά διαθέσιμο. Η επιλογή `random_state=42` εξασφαλίζει ότι η ίδια τυχαία δειγματοληψία μπορεί να αναπαραχθεί σε επόμενες εκτελέσεις του κώδικα, γεγονός ιδιαίτερα σημαντικό για τη συνέπεια των πειραμάτων. Τέλος, η μεταβλητή `evaluated_users` αρχικοποιείται στο μηδέν και θα χρησιμοποιηθεί αργότερα για να καταγράψει πόσοι χρήστες αξιολογήθηκαν πράγματι, αφού

ενδέχεται ορισμένοι να παραλειφθούν στα επόμενα στάδια αν δεν διαθέτουν relevant αντικείμενα ή αν δεν μπορούν να παραχθούν προτάσεις για αυτούς.

```

364     for user_id in sample_users:
365         relevant_items = set(
366             test[(test["userId"] == user_id) & (test["rating"] >= eval_threshold)]["movieId"].unique()
367         )
368         if len(relevant_items) == 0:
369             continue
370         recs = recommend_for_user(user_id, top_k=top_k, use_popularity=use_popularity)
371         if recs.empty:
372             continue
373
374         recommended_items = recs["movieId"].tolist()
375         recommended_scores = recs["score"].tolist()
376
377         precision_scores.append(precision_at_k(recommended_items, relevant_items, top_k))
378         recall_scores.append(recall_at_k(recommended_items, relevant_items, top_k))
379         map_scores.append(average_precision_at_k(recommended_items, relevant_items, top_k))
380         ndcg_scores_list.append(ndcg_at_k(recommended_items, recommended_scores, relevant_items, top_k))
381         hitrate_scores.append(hit_rate_at_k(recommended_items, relevant_items, top_k))
382         diversity_scores.append(intra_list_diversity(recommended_items))
383
384         evaluated_users += 1
385
386     return {
387         "Users evaluated": evaluated_users,
388         f"Precision@{top_k}": np.mean(precision_scores) if precision_scores else 0.0,
389         f"Recall@{top_k}": np.mean(recall_scores) if recall_scores else 0.0,
390         f"MAP@{top_k}": np.mean(map_scores) if map_scores else 0.0,
391         f"NDCG@{top_k}": np.mean(ndcg_scores_list) if ndcg_scores_list else 0.0,
392         f"HitRate@{top_k}": np.mean(hitrate_scores) if hitrate_scores else 0.0,
393         "Diversity": np.mean(diversity_scores) if diversity_scores else 0.0
394     }

```

Εικόνα 43: Υπολογισμός μετρικών αξιολόγησης

Στο κύριο μέρος της συνάρτησης, η αξιολόγηση πραγματοποιείται επαναληπτικά για κάθε χρήστη του δείγματος `sample_users`. Για κάθε `user_id`, ο κώδικας εντοπίζει πρώτα τα `relevant_items`, δηλαδή τις ταινίες του test set που ανήκουν στον συγκεκριμένο χρήστη και έχουν βαθμολογία μεγαλύτερη ή ίση από το `eval_threshold`. Οι ταινίες αυτές μετατρέπονται σε σύνολο (set), ώστε να είναι εύκολος και αποδοτικός ο έλεγχος του αν μία προτεινόμενη ταινία θεωρείται σχετική. Αν ο χρήστης δεν διαθέτει κανένα relevant αντικείμενο, παραλείπεται με την εντολή `continue`, καθώς σε αυτή την περίπτωση δεν έχει νόημα να υπολογιστούν μετρικές όπως Precision, Recall ή MAP.

Στη συνέχεια, για τον ίδιο χρήστη καλείται η `recommend_for_user`, η οποία παράγει τη λίστα προτάσεων με βάση το περιεχόμενο και, προαιρετικά, τη δημοφιλία. Αν η συνάρτηση επιστρέψει κενό DataFrame, ο χρήστης παραλείπεται επίσης, αφού δεν υπάρχουν διαθέσιμες

προτάσεις για να αξιολογηθούν. Όταν υπάρχουν έγκυρες προτάσεις, ο κώδικας εξάγει σε λίστες τα `movieId` των προτεινόμενων ταινιών (`recommended_items`) και τα αντίστοιχα σκορ τους (`recommended_scores`). Οι δύο αυτές λίστες αποτελούν την είσοδο για τον υπολογισμό των μετρικών αξιολόγησης.

Στο επόμενο βήμα, για κάθε χρήστη υπολογίζονται και αποθηκεύονται οι επιμέρους τιμές των μετρικών. Η `precision_at_k` μετρά το ποσοστό των προτεινόμενων ταινιών που είναι σχετικές, η `recall_at_k` το ποσοστό των `relevant` ταινιών που ανακτήθηκαν από το σύστημα, η `average_precision_at_k` αποτιμά τη μέση ακρίβεια της κατάταξης λαμβάνοντας υπόψη τη θέση των `relevant` αντικειμένων, και η `ndcg_at_k` αξιολογεί την ποιότητα της διάταξης των προτάσεων με μεγαλύτερη έμφαση στις υψηλότερες θέσεις. Παράλληλα, η `hit_rate_at_k` ελέγχει αν υπάρχει τουλάχιστον μία σχετική ταινία μέσα στις πρώτες `k` προτάσεις, ενώ η `intra_list_diversity` υπολογίζει την εσωτερική ποικιλία της λίστας. Κάθε τιμή προστίθεται στη αντίστοιχη λίστα μετρικών και, όταν ολοκληρωθεί επιτυχώς η αξιολόγηση ενός χρήστη, αυξάνεται κατά ένα η μεταβλητή `evaluated_users`.

Εν συνεχεία, η συνάρτηση επιστρέφει ένα λεξικό με τα συνολικά αποτελέσματα της αξιολόγησης. Το πεδίο "`Users evaluated`" καταγράφει το πλήθος των χρηστών που αξιολογήθηκαν πράγματι, δηλαδή όσων δεν παραλείφθηκαν στα προηγούμενα βήματα. Για κάθε μετρική υπολογίζεται ο μέσος όρος των επιμέρους τιμών που έχουν αποθηκευτεί στις αντίστοιχες λίστες, μέσω της `np.mean`. Αν κάποια λίστα είναι κενή, επιστρέφεται τιμή 0.0, ώστε να αποφεύγονται σφάλματα και να εξασφαλίζεται σταθερή συμπεριφορά της συνάρτησης. Με αυτόν τον τρόπο, η `evaluate_model` δεν επιστρέφει τις μετρικές κάθε χρήστη ξεχωριστά, αλλά μια συνολική εικόνα της απόδοσης του recommender πάνω στο επιλεγμένο δείγμα χρηστών.

```

399 def print_user_history(user_id, max_rows=15): 1 usage new *
400     user_train = user_train_dict.get(user_id)
401
402     if user_train is None or user_train.empty:
403         print(f"\nNo train history found for user {user_id}")
404         return
405
406     user_seen = user_train[
407         (user_train["movieId"].isin(movie_id_to_idx)) &
408         (user_train["rating"] >= PROFILE_THRESHOLD)
409     ].copy()
410
411     if user_seen.empty:
412         print(f"\nNo usable positive history found for user {user_id}")
413         return
414
415     user_seen = user_seen.merge(
416         movies[["movieId", "title", "genres"]],
417         on="movieId",
418         how="left"
419     )
420
421     user_seen = user_seen.sort_values(by=["rating"], ascending=False).head(max_rows)
422
423     print(f"\nTop positively rated movies already seen by user {user_id}:")
424     print(user_seen[["movieId", "title", "rating", "genres"]].reset_index(drop=True))

```

Εικόνα 44: Ιστορικό θετικών αξιολογήσεων χρήστη

Η συνάρτηση `print_user_history` χρησιμοποιείται για την εμφάνιση του θετικού ιστορικού ενός χρήστη από το training set, προσφέροντας μια πιο κατανοητή εικόνα των ταινιών πάνω στις οποίες βασίζεται το προφίλ του. Αρχικά, ο κώδικας ανακτά από το λεξικό `user_train_dict` το υποσύνολο του train που αντιστοιχεί στον συγκεκριμένο χρήστη. Αν δεν βρεθεί ιστορικό ή αν αυτό είναι κενό, η συνάρτηση εμφανίζει σχετικό μήνυμα και τερματίζει, καθώς δεν υπάρχουν διαθέσιμα δεδομένα για παρουσίαση.

Ως επόμενο βήμα, από το ιστορικό του χρήστη διατηρούνται μόνο οι ταινίες που υπάρχουν στο `movie_id_to_idx` και έχουν βαθμολογία μεγαλύτερη ή ίση από το `PROFILE_THRESHOLD`. Έτσι, η συνάρτηση δεν εμφανίζει όλες τις αξιολογημένες ταινίες του χρήστη, αλλά μόνο εκείνες που θεωρούνται θετικές και μπορούν πράγματι να αξιοποιηθούν από το μοντέλο για τη δημιουργία του προφίλ του. Αν και μετά από αυτό το φιλτράρισμα δεν απομένουν εγγραφές, εμφανίζεται νέο μήνυμα και η συνάρτηση τερματίζει, δηλώνοντας ότι δεν υπάρχει αξιοποιήσιμο θετικό ιστορικό.

Μετάπειτα, τα επιλεγμένα δεδομένα συγχωνεύονται με τον πίνακα *movies*, ώστε για κάθε *movieId* να προστεθούν ο τίτλος και τα *genres* της αντίστοιχης ταινίας. Με αυτόν τον τρόπο, το ιστορικό γίνεται πιο αναγνώσιμο και δεν περιορίζεται μόνο σε αριθμητικά αναγνωριστικά. Στη συνέχεια, οι εγγραφές ταξινομούνται κατά φθίνουσα σειρά ως προς τη βαθμολογία και διατηρούνται μόνο οι πρώτες *max_rows*, ώστε η εμφάνιση να περιορίζεται στις σημαντικότερες θετικά αξιολογημένες ταινίες. Τέλος, η συνάρτηση εκτυπώνει ένα σύντομο ενημερωτικό μήνυμα και στη συνέχεια παρουσιάζει έναν πίνακα με τα πεδία *movieId*, *title*, *rating* και *genres*, προσφέροντας έτσι μια συνοπτική αλλά ουσιαστική εικόνα του θετικού ιστορικού του χρήστη.

```
429 > if __name__ == "__main__":
430     random_user = np.random.choice(train["userId"].unique())
431
432     print_user_history(random_user, max_rows=15)
433
434     print(f"\nRecommendations for user {random_user}:")
435     print(recommend_for_user(random_user, top_k=TOP_K, use_popularity=USE_POPULARITY))
436
437     print("\nEvaluating content-based recommender...")
438     results = evaluate_model(
439         sample_size=EVAL_SAMPLE_USERS,
440         top_k=TOP_K,
441         eval_threshold=EVAL_THRESHOLD,
442         use_popularity=USE_POPULARITY
443     )
444
445     print("\nEvaluation Results:")
446     for metric, value in results.items():
447         if isinstance(value, float):
448             print(f"{metric}: {value:.4f}")
449         else:
450             print(f"{metric}: {value}")
```

Εικόνα 45: Εκτέλεση και αξιολόγηση του μοντέλου συστάσεων

Στο τελικό τμήμα του προγράμματος ορίζεται η κύρια ροή εκτέλεσης του συστήματος. Η συνθήκη `if __name__ == "__main__":` εξασφαλίζει ότι το συγκεκριμένο μέρος του κώδικα θα εκτελεστεί μόνο όταν το αρχείο τρέχει ως κύριο πρόγραμμα και όχι όταν εισάγεται ως *module* σε άλλο *script*. Μέσα σε αυτό το πλαίσιο, επιλέγεται αρχικά ένας τυχαίος χρήστης από το *training set* με χρήση της `np.random.choice(train["userId"].unique())`, ώστε να παρουσιαστεί ένα ενδεικτικό παράδειγμα λειτουργίας του recommender πάνω σε πραγματικό ιστορικό χρήστη.

Ακόμη, καλείται η `print_user_history(random_user, max_rows=15)`, η οποία εμφανίζει έως 15 από τις θετικά αξιολογημένες ταινίες του συγκεκριμένου χρήστη. Με αυτόν τον τρόπο παρέχεται πρώτα μια εικόνα του ιστορικού πάνω στο οποίο βασίζεται το προφίλ του, ώστε να είναι πιο εύκολη η ερμηνεία των προτάσεων που θα ακολουθήσουν. Αμέσως μετά, ο κώδικας εμφανίζει ένα ενημερωτικό μήνυμα και καλεί τη `recommend_for_user`, περνώντας ως παραμέτρους τον τυχαίο χρήστη, το πλήθος προτάσεων `TOP_K` και τη ρύθμιση `USE_POPULARITY`. Το αποτέλεσμα που εκτυπώνεται είναι η τελική λίστα προτεινόμενων ταινιών για τον συγκεκριμένο χρήστη, μαζί με το αντίστοιχο score κάθε πρότασης.

Έπειτα, το πρόγραμμα περνά στο στάδιο της συνολικής αξιολόγησης του συστήματος. Εμφανίζεται σχετικό μήνυμα και καλείται η συνάρτηση `evaluate_model`, στην οποία δίνονται ως παράμετροι το μέγεθος του δείγματος χρηστών (`EVAL_SAMPLE_USERS`), το πλήθος προτάσεων ανά χρήστη (`TOP_K`), το κατώφλι σχετικότητας για το test set (`EVAL_THRESHOLD`) και η επιλογή χρήσης δημοφιλίας (`USE_POPULARITY`). Η συνάρτηση επιστρέφει ένα λεξικό `results` που περιέχει τις τελικές συγκεντρωτικές μετρικές απόδοσης του recommender πάνω στο επιλεγμένο δείγμα χρηστών. Εν κατακλείδι, ο κώδικας εκτυπώνει τα αποτελέσματα της αξιολόγησης. Για κάθε στοιχείο του λεξικού `results`, ελέγχεται αν η τιμή είναι δεκαδικός αριθμός· αν ναι, εμφανίζεται μορφοποιημένη με τέσσερα δεκαδικά ψηφία, ενώ διαφορετικά εκτυπώνεται αυτούσια. Με αυτόν τον τρόπο, το τελικό στάδιο του προγράμματος συνδυάζει δύο λειτουργίες: αφενός παρουσιάζει ένα ενδεικτικό παράδειγμα προτάσεων για έναν τυχαίο χρήστη και αφετέρου παρέχει μια συνολική εικόνα της απόδοσης του μοντέλου μέσω των μετρικών αξιολόγησης.

4.1.3 Παρουσίαση των αποτελεσμάτων αρχείου content-based.py

```
Movies raw: (87585, 3)
Tags raw: (2000072, 4)
Train raw: (25474164, 6)
Test raw: (6368541, 6)
TMDb raw: (31478, 21)
Unique movies in original movies.csv: 87585
Unique movies in TRAIN: 31961
Unique movies in TEST: 31445
Candidate movies for recommendation: 31961
Tags after cleaning: (1650095, 4)
TF-IDF matrix: (31961, 76282)
Train after alignment: (25474164, 6)
Test after alignment: (6368541, 6)
Final candidate movie count: 31961
```

Εικόνα 46: Αποτελέσματα προεπεξεργασίας και TF-IDF αναπαράστασης

Στο πρώτο στάδιο εκτέλεσης του κώδικα επιβεβαιώνεται το αρχικό μέγεθος των βασικών πηγών δεδομένων. Ο πίνακας movies περιέχει 87.585 ταινίες και 3 βασικές στήλες, ο πίνακας tags περιλαμβάνει 2.000.072 εγγραφές, ενώ τα σύνολα train και test περιέχουν 25.474.164 και 6.368.541 αξιολογήσεις αντίστοιχα. Παράλληλα, το αρχείο εμπλουτισμού από το TMDb περιλαμβάνει 31.478 εγγραφές. Τα μεγέθη αυτά δείχνουν ότι το σύστημα βασίζεται σε πολύ μεγάλο όγκο δεδομένων, τόσο ως προς τις αξιολογήσεις όσο και ως προς τα συνοδευτικά μεταδεδομένα των ταινιών.

Σε αυτό το σημείο είναι σημαντικό να διευκρινιστεί ότι το movies.csv λειτουργεί ως ο πλήρης αρχικός κατάλογος ταινιών του dataset, ενώ τα αρχεία train και test δεν αποτελούν διαχωρισμό του ίδιου του καταλόγου ταινιών, αλλά διαχωρισμό των αξιολογήσεων των χρηστών. Επομένως, από τα train και test δεν προκύπτουν δύο ξεχωριστοί κατάλογοι ταινιών με άμεσο τρόπο, αλλά δύο σύνολα αξιολογήσεων μέσα από τα οποία μπορούμε να εντοπίσουμε ποιες ταινίες εμφανίζονται πράγματι σε κάθε split. Για αυτόν τον λόγο, ενώ το αρχικό movies.csv περιλαμβάνει 87.585 μοναδικές ταινίες, μόνο 31.961 από αυτές εμφανίζονται τουλάχιστον μία φορά στο training set και 31.445 εμφανίζονται στο test set.

Με βάση τη σχεδιαστική επιλογή του μοντέλου, ο τελικός candidate catalog δεν ταυτίζεται με το πλήρες movies.csv, αλλά περιορίζεται αποκλειστικά στις ταινίες που εμφανίζονται στο training set. Αυτό υλοποιείται με το φιλτράρισμα του πίνακα movies ώστε να διατηρηθούν μόνο τα movieId που ανήκουν στο train_item_ids. Ως αποτέλεσμα, από τις 87.585 ταινίες του αρχικού καταλόγου παραμένουν τελικά 31.961 υποψήφιες ταινίες προς σύσταση, δηλαδή ακριβώς όσες εμφανίζονται στο train. Η επιλογή αυτή σημαίνει ότι το σύστημα προτείνει μόνο αντικείμενα για τα οποία διαθέτει πληροφορία κατά το στάδιο της εκπαίδευσης και της κατασκευής των χαρακτηριστικών, αποφεύγοντας έτσι την παραγωγή προτάσεων για ταινίες που υπάρχουν μεν στο αρχικό dataset αλλά δεν έχουν ενταχθεί ουσιαστικά στον χώρο αναπαράστασης του μοντέλου.

Ιδιαίτερο ενδιαφέρον παρουσιάζει και το στάδιο καθαρισμού των tags. Από τις αρχικές 2.000.072 εγγραφές, μετά το φιλτράρισμα παραμένουν 1.650.095. Η μείωση αυτή δείχνει ότι αφαιρέθηκε σημαντικός όγκος θορυβώδους ή χαμηλής χρησιμότητας πληροφορίας, όπως πολύ σπάνια tags, πολύ σύντομα tags ή τιμές χωρίς ουσιαστικό περιγραφικό περιεχόμενο. Παρ' όλα αυτά, το τελικό πλήθος των tags παραμένει ιδιαίτερα υψηλό, γεγονός που δείχνει ότι τα tags εξακολουθούν να αποτελούν σημαντική πηγή περιγραφικής πληροφορίας για το μοντέλο.

Η δημιουργία της TF-IDF μήτρας οδηγεί σε αναπαράσταση διαστάσεων (31961, 76282), δηλαδή 31.961 ταινίες και 76.282 λεκτικά χαρακτηριστικά. Το αποτέλεσμα αυτό δείχνει ότι, μετά τον καθαρισμό και τη σύνθεση του rich_content, το μοντέλο καταλήγει σε ένα ιδιαίτερα πλούσιο λεξιλόγιο χαρακτηριστικών, ικανό να αποτυπώσει πολυδιάστατα το περιεχόμενο των ταινιών. Παράλληλα, το γεγονός ότι το πλήθος των γραμμών της μήτρας ταυτίζεται με το πλήθος του candidate catalog επιβεβαιώνει ότι κάθε ταινία που παραμένει υποψήφια για σύσταση διαθέτει τελικά πλήρη διανυσματική αναπαράσταση.

Σε αυτό το σημείο, μετά την ευθυγράμμιση των train και test με το τελικό λεξικό movie_id_to_idx, τα μεγέθη τους παραμένουν αμετάβλητα. Αυτό σημαίνει ότι όλες οι αξιολογήσεις που είχαν διατηρηθεί στα δύο σύνολα αντιστοιχούν ήδη σε ταινίες του τελικού candidate catalog. Με άλλα λόγια, παρότι το test set αποτελεί ξεχωριστό σύνολο αξιολογήσεων, στην τελική του μορφή αξιολογεί μόνο ταινίες που υπάρχουν και στον χώρο των υποψήφιων προτάσεων του μοντέλου. Συνολικά, τα αποτελέσματα αυτού του σταδίου δείχνουν ότι η διαδικασία περιορισμού του catalog, καθαρισμού των δεδομένων και διανυσματοποίησης ολοκληρώθηκε με συνέπεια, οδηγώντας σε ένα μεγάλο αλλά πλήρως

ευθυγραμμισμένο σύνολο δεδομένων πάνω στο οποίο μπορεί να εφαρμοστεί με αξιοπιστία το content-based recommender.

Top positively rated movies already seen by user 77546:

	movieId	...	genres
0	2872	...	Adventure Fantasy
1	1580	...	Action Comedy Sci-Fi
2	2161	...	Adventure Children Fantasy
3	1073	...	Children Comedy Fantasy Musical
4	3466	...	Comedy Fantasy
5	2174	...	Comedy Fantasy
6	653	...	Action Adventure Fantasy
7	2384	...	Adventure Children Drama
8	3175	...	Adventure Comedy Sci-Fi
9	480	...	Action Adventure Sci-Fi Thriller
10	2193	...	Action Adventure Fantasy
11	2628	...	Action Adventure Sci-Fi

[12 rows x 4 columns]

Εικόνα 47: Θετικά αξιολογημένες ταινίες χρήστη

Το εμφανιζόμενο αποτέλεσμα παρουσιάζει το θετικό ιστορικό του χρήστη 77546, δηλαδή τις ταινίες του training set που έχουν αξιολογηθεί με βαθμολογία ίση ή μεγαλύτερη από το κατώφλι PROFILE_THRESHOLD και συνεπώς χρησιμοποιούνται για τη δημιουργία του προφίλ προτιμήσεών του. Παρατηρείται ότι το ιστορικό αυτό δεν είναι τυχαίο, αλλά εμφανίζει σαφή θεματική συνοχή, καθώς κυριαρχούν genres όπως Adventure, Fantasy, Action, Comedy και Sci-Fi, ενώ σε μικρότερο βαθμό εμφανίζονται και στοιχεία Children, Drama, Musical και Thriller. Η επαναλαμβανόμενη παρουσία αυτών των κατηγοριών δείχνει ότι ο συγκεκριμένος χρήστης έχει σταθερή προτίμηση σε ταινίες με περιπετειώδη, φανταστικό και ψυχαγωγικό χαρακτήρα, συχνά συνδυασμένο με δράση ή επιστημονική φαντασία. Το στοιχείο αυτό είναι ιδιαίτερα σημαντικό για τη λειτουργία του content-based recommender, καθώς το μοντέλο κατασκευάζει το προφίλ του χρήστη ως σταθμισμένο συνδυασμό των χαρακτηριστικών των θετικά αξιολογημένων ταινιών του. Επομένως, όταν συγκεκριμένα

genres εμφανίζονται επανειλημμένα στο ιστορικό, τα αντίστοιχα λεκτικά χαρακτηριστικά αποκτούν μεγαλύτερη βαρύτητα στο τελικό user profile.

Η εικόνα αυτή λειτουργεί ως ποιοτική επιβεβαίωση ότι το σύστημα διαθέτει επαρκές και σχετικά συνεκτικό ιστορικό για να σχηματίσει ουσιαστικό προφίλ προτιμήσεων για τον συγκεκριμένο χρήστη. Με άλλα λόγια, το μοντέλο δεν βασίζεται σε λίγες αποσπασματικές αξιολογήσεις, αλλά σε ένα σύνολο ταινιών που παρουσιάζει σαφή κοινά χαρακτηριστικά. Αυτό αυξάνει τη δυνατότητα του recommender να εντοπίσει νέες ταινίες με παρόμοιο περιεχόμενο και να παράγει περισσότερο στοχευμένες προτάσεις. Συνεπώς, πριν ακόμη εξεταστούν οι τελικές συστάσεις, το θετικό ιστορικό του χρήστη υποδηλώνει ότι η διαδικασία δημιουργίας προφίλ στηρίζεται σε επαρκή πληροφορία και ότι οι επόμενες προτάσεις αναμένεται λογικά να κινηθούν γύρω από τον ίδιο θεματικό άξονα, δηλαδή ταινίες με έμφαση στην περιπέτεια, τη φαντασία, τη δράση και την επιστημονική φαντασία.

```
Recommendations for user 77546:
```

	movieId	...	score
0	5459	...	0.334659
1	260	...	0.300245
2	5378	...	0.295271
3	1210	...	0.287928
4	1196	...	0.282132
5	33493	...	0.281126
6	5952	...	0.275543
7	7153	...	0.273326
8	1587	...	0.269658
9	41566	...	0.259371

[10 rows x 4 columns]

Evaluating content-based recommender...

Evaluation Results:

Users evaluated: 293

Precision@10: 0.1911

Recall@10: 0.1251

MAP@10: 0.1470

NDCG@10: 0.5094

HitRate@10: 0.7099

Diversity: 0.8677

Εικόνα 48: Συστάσεις και αξιολόγηση content-based μοντέλου

Το παραγόμενο σύνολο προτάσεων για τον χρήστη 77546 είναι συνεπές με το θετικό ιστορικό που είχε προηγουμένως παρουσιαστεί. Οι δέκα ταινίες που προτείνονται αποτελούν τις κορυφαίες επιλογές του μοντέλου με βάση το τελικό score ομοιότητας, το οποίο στην παρούσα υλοποίηση προκύπτει κυρίως από την cosine similarity ανάμεσα στο προφίλ του χρήστη και στα TF-IDF διανύσματα των υποψήφιων ταινιών, με μικρή επιρροή και από τον παράγοντα δημοφιλίας. Οι τιμές score κυμαίνονται από 0.2594 έως 0.3347, γεγονός που

δείχνει ότι το μοντέλο εντόπισε ένα σύνολο ταινιών με σχετικά υψηλή συνάφεια ως προς τα χαρακτηριστικά που έχουν εξαχθεί από το περιεχόμενο. Ερμηνευτικά, το αποτέλεσμα αυτό επιβεβαιώνει ότι το προφίλ του χρήστη ήταν αρκετά σαφές ώστε να οδηγήσει σε στοχευμένες προτάσεις και ότι ο μηχανισμός κατάταξης είναι σε θέση να ξεχωρίσει εκείνα τα αντικείμενα που βρίσκονται πιο κοντά στις εκπεφρασμένες προτιμήσεις του.

Σε επίπεδο συνολικής αξιολόγησης, το μοντέλο εξετάστηκε σε 293 χρήστες, δηλαδή σε όσους παρέμειναν τελικά κατάλληλοι για αξιολόγηση μετά τους σχετικούς ελέγχους του pipeline. Για το συγκεκριμένο πείραμα προέκυψαν οι τιμές $\text{Precision@10} = 0.1911$, $\text{Recall@10} = 0.1251$, $\text{MAP@10} = 0.1470$, $\text{NDCG@10} = 0.5094$, $\text{HitRate@10} = 0.7099$ και $\text{Diversity} = 0.8677$. Οι τιμές αυτές αποτυπώνουν τη συνολική απόδοση του content-based recommender στο επιλεγμένο δείγμα χρηστών και παρέχουν μια συγκεντρωτική εικόνα της συμπεριφοράς του μοντέλου ως προς τη συνάφεια, την ποιότητα κατάταξης και την ποικιλία των προτάσεων.

4.1.4 Αξιολόγηση Μεθόδου

Στο τελευταίο στάδιο της εκτέλεσης πραγματοποιείται η αξιολόγηση του συστήματος προτάσεων σε τυχαίο δείγμα χρηστών από το test set. Για τη συγκεκριμένη εκτέλεση χρησιμοποιήθηκε δείγμα 300 χρηστών, από τους οποίους τελικά αξιολογήθηκαν 293, καθώς κάποιοι χρήστες δεν διέθεταν relevant αντικείμενα στο test set ή δεν ήταν δυνατό να παραχθούν έγκυρες προτάσεις για αυτούς. Για κάθε χρήστη το σύστημα παράγει τις τοπ 10 προτάσεις και στη συνέχεια τις συγκρίνει με τις relevant ταινίες του test set, δηλαδή με τις ταινίες που έχουν βαθμολογία μεγαλύτερη ή ίση από το κατώφλι αξιολόγησης. Με βάση αυτή τη διαδικασία υπολογίζονται οι μετρικές Precision@10 , Recall@10 , MAP@10 , NDCG@10 , HitRate@10 και Diversity , ενώ στο τέλος εμφανίζεται ο μέσος όρος τους για το σύνολο των αξιολογημένων χρηστών. Για τη συγκεκριμένη εκτέλεση προέκυψαν οι εξής τιμές:

- $\text{Precision@10} = 0.1911$
- $\text{Recall@10} = 0.1251$
- $\text{MAP@10} = 0.1470$
- $\text{NDCG@10} = 0.5094$
- $\text{HitRate@10} = 0.7099$
- $\text{Diversity} = 0.8677$

Πιο συγκεκριμένα, η τιμή του Precision@10 δείχνει τι ποσοστό από τις δέκα προτάσεις του συστήματος είναι πράγματι σχετικό με τις προτιμήσεις του χρήστη. Στη συγκεκριμένη περίπτωση, η τιμή 0.1911 σημαίνει ότι κατά μέσο όρο περίπου 1,9 από τις 10 προτάσεις είναι relevant. Επομένως, η μετρική αυτή δείχνει πόσο «καθαρή» είναι η λίστα προτάσεων, δηλαδή πόσες από τις πρώτες θέσεις καταλαμβάνονται από σωστές επιλογές. Η τιμή του Recall@10 δείχνει κάτι διαφορετικό: δεν μετρά πόσες από τις προτάσεις ήταν σωστές, αλλά πόσο μεγάλο μέρος των συνολικά relevant ταινιών κατάφερε να εντοπίσει το σύστημα μέσα στις πρώτες δέκα θέσεις. Εδώ η τιμή 0.1251 σημαίνει ότι το μοντέλο ανακτά περίπου το 12,5% των relevant αντικειμένων του χρήστη στο top-10. Με άλλα λόγια, το Precision δείχνει πόσο στοχευμένες είναι οι προτάσεις, ενώ το Recall δείχνει πόσο καλά καλύπτεται το σύνολο των σωστών επιλογών.

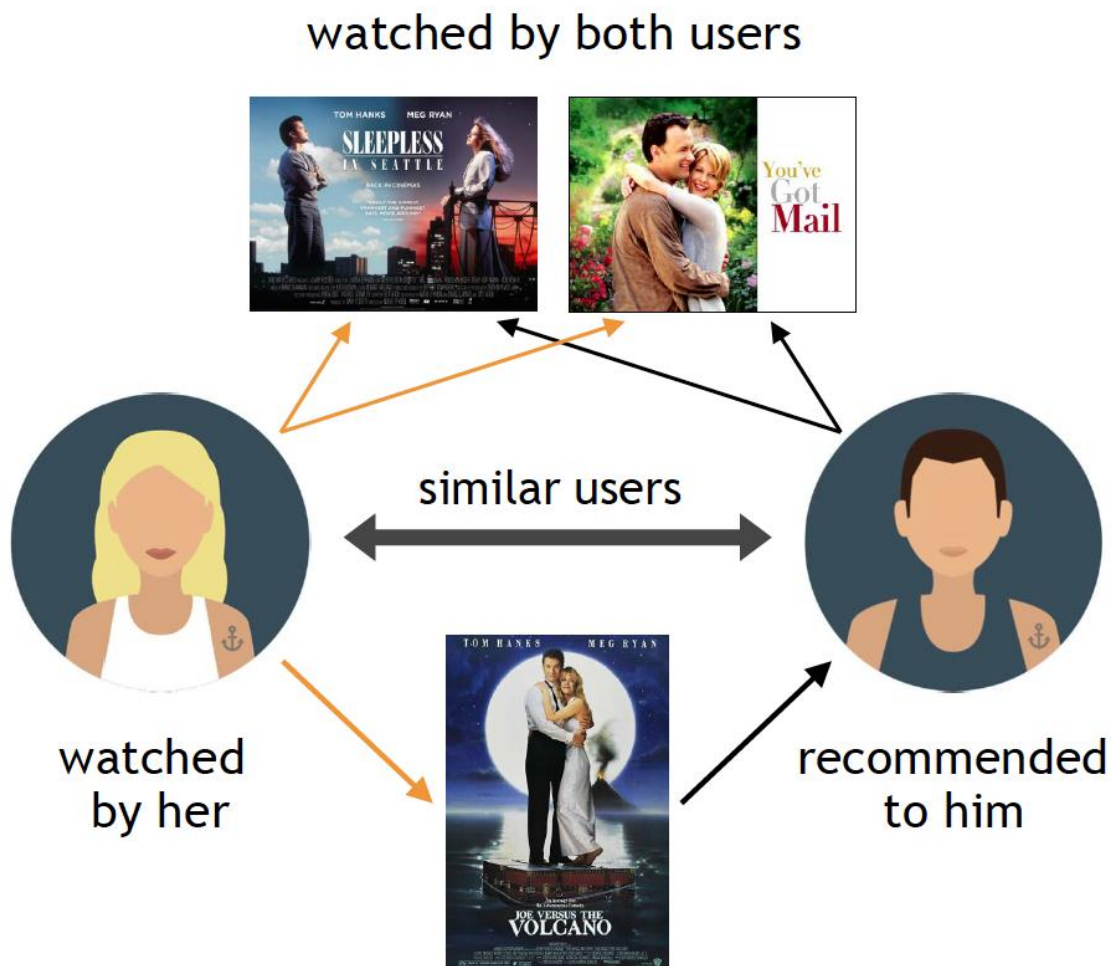
Η μετρική MAP@10 = 0.1470 δείχνει τη μέση ακρίβεια της λίστας προτάσεων, λαμβάνοντας υπόψη και τη σειρά με την οποία εμφανίζονται οι relevant ταινίες. Αυτό σημαίνει ότι δεν αρκεί απλώς να βρεθεί μια σωστή ταινία, αλλά έχει σημασία και αν εμφανίζεται νωρίς στη λίστα ή χαμηλότερα. Άρα η συγκεκριμένη μετρική αποτιμά πιο ουσιαστικά την ποιότητα της κατάταξης. Αντίστοιχα, η NDCG@10 = 0.5094 δείχνει επίσης την ποιότητα της κατάταξης, αλλά με τρόπο που δίνει μεγαλύτερη σημασία στις πρώτες θέσεις της λίστας. Η τιμή αυτή δείχνει ότι το μοντέλο καταφέρνει σε αρκετές περιπτώσεις να τοποθετεί relevant ταινίες σχετικά ψηλά στις προτάσεις του. Η HitRate@10 = 0.7099 είναι πιο απλή στην ερμηνεία της και ιδιαίτερα χρήσιμη στην πράξη, καθώς δείχνει σε πόσους χρήστες το σύστημα κατάφερε να δώσει έστω και μία σχετική πρόταση μέσα στις πρώτες δέκα θέσεις. Στην προκειμένη περίπτωση, η τιμή αυτή σημαίνει ότι περίπου στο 71% των χρηστών το σύστημα πέτυχε τουλάχιστον ένα «χτύπημα», δηλαδή τουλάχιστον μία χρήσιμη πρόταση. Τέλος, η Diversity = 0.8677 δείχνει τον βαθμό ποικιλίας των ταινιών μέσα στην ίδια λίστα προτάσεων. Η υψηλή αυτή τιμή σημαίνει ότι οι προτεινόμενες ταινίες δεν είναι υπερβολικά όμοιες μεταξύ τους, αλλά παρουσιάζουν ικανοποιητική εσωτερική διαφοροποίηση, κάτι που είναι σημαντικό ώστε το σύστημα να μην προτείνει συνεχώς σχεδόν το ίδιο είδος αντικειμένου.

Συνολικά, τα αποτελέσματα δείχνουν ότι το μοντέλο έχει ικανοποιητική συμπεριφορά για content-based σύστημα, καθώς καταφέρνει να εντοπίζει relevant ταινίες, να τις εμφανίζει σε αρκετές περιπτώσεις μέσα στις πρώτες θέσεις και παράλληλα να διατηρεί αρκετά υψηλή ποικιλία στις λίστες προτάσεων. Η αξιολόγηση αυτή είναι αρκετά αυστηρή, καθώς μια relevant

ταινία μετρά θετικά μόνο όταν το μοντέλο καταφέρει να την τοποθετήσει μέσα στο top-10. Συνεπώς, relevant ταινίες που βρίσκονται χαμηλότερα στην κατάταξη δεν αποτυπώνονται θετικά στις μετρικές, ακόμη και αν είναι πράγματι σχετικές με τις προτιμήσεις του χρήστη. Επιπλέον, το σύστημα προτείνει μόνο από το candidate catalog του training set, επομένως οι μετρικές εκφράζουν την απόδοσή του μέσα σε αυτό το συγκεκριμένο recommendation space. Παρόλα αυτά, η συνολική εικόνα δείχνει ότι το μοντέλο επιτυγχάνει μια καλή ισορροπία ανάμεσα στη συνάφεια, την κατάταξη και την ποικιλία των προτάσεων.

4.2 Collaborative Filtering

Collaborative Filtering



Εικόνα 49: Collaborative Filtering, Πηγή: nvidia.com

Το συνεργατικό φιλτράρισμα αποτελεί μία από τις πλέον διαδεδομένες και μελετημένες προσεγγίσεις στο χώρο των συστημάτων προτάσεων, καθώς βασίζεται στη συλλογική συμπεριφορά των χρηστών και όχι στα εγγενή χαρακτηριστικά των αντικειμένων (Gazdar & Hidri, 2020). Η βασική φιλοσοφία της μεθόδου εδράζεται στην υπόθεση ότι χρήστες που παρουσίασαν παρόμοιες προτιμήσεις στο παρελθόν είναι πιθανό να εμφανίσουν παρόμοια συμπεριφορά και στο μέλλον, γεγονός που επιτρέπει την εξαγωγή εξατομικευμένων προτάσεων βάσει ιστορικών αλληλεπιδράσεων. Σε αντίθεση με τις μεθόδους φιλτραρίσματος με βάση το περιεχόμενο (Content-based filtering), που εξετάσαμε προηγουμένως, οι οποίες

απαιτούν μεταδεδομένα για κάθε αντικείμενο, το συνεργατικό φιλτράρισμα αξιοποιεί αποκλειστικά δεδομένα αλληλεπίδρασης χρηστών – αντικειμένων, όπως ρητές αξιολογήσεις ή έμμεσες συμπεριφορές χρήσης (Martins et al., 2020). Η ανεξαρτησία της μεθόδου από τα χαρακτηριστικά του περιεχομένου την καθιστά ιδιαίτερα ελκυστική σε περιβάλλοντα όπου η περιγραφή των αντικειμένων ελλιπής, ασαφής και δύσκολη (Fkih, 2022).

Στο πλαίσιο των συστημάτων προτάσεων, το Collaborative Filtering διατυπώνεται συχνά ως πρόβλημα συμπλήρωσης αραιών πινάκων (sparse matrix completion), όπου οι γραμμές αντιστοιχούν σε χρήστες, οι στήλες σε αντικείμενα και οι τιμές σε αξιολογήσεις ή εντάσεις αλληλεπίδρασης. Στην πράξη, μόνο ένα μικρό ποσοστό των δυνατών αλληλεπιδράσεων παρατηρείται, με αποτέλεσμα ο πίνακας χρηστών – αντικειμένων να χαρακτηρίζεται από υψηλή αραιότητα, η οποία αποτελεί μία από τις βασικότερες προκλήσεις του συνεργατικού φιλτραρίσματος (Martins et al., 2020).

Για την αντιμετώπιση των παραπάνω περιορισμών, η έρευνα στον χώρο του Collaborative Filtering έχει στραφεί σε πιο εξελιγμένες μοντελοκεντρικές (model-based) προσεγγίσεις, οι οποίες βασίζονται στην μάθηση παραμέτρων από τα δεδομένα και όχι στον άμεσο υπολογισμό ομοιότητας μεταξύ χρηστών ή αντικειμένων (Martins et al., 2020). Οι προσεγγίσεις αυτές επιχειρούν να αναπαραστήσουν χρήστες και αντικείμενα μέσω λανθανόντων παραγόντων σε έναν χαμηλότερης διάστασης χώρο, συνοψίζοντας την υποκειμενική δομή των προτιμήσεων και μειώνοντας την επίδραση της αραιότητας. Στο πλαίσιο των model – based μεθόδων, η παραγοντοποίηση πινάκων (matrix – factorization) αποτελεί μία από τις πιο επιτυχημένες και ευρέως υιοθετημένες τεχνικές, καθώς επιτρέπει την αποδοτική χαρτογράφηση λανθανόντων αναπαραστάσεων χρηστών και αντικειμένων (latent vectors), σε έναν κοινό χώρο χαμηλότερης διάστασης (Papadakis et al., 2022).

Πιο συγκεκριμένα, ο αλγόριθμος Alternating Least Squares (ALS) αποτελεί καθιερωμένη τεχνική παραγοντοποίησης πινάκων, στο πλαίσιο των model-based collaborative filtering μεθόδων, η οποία χρησιμοποιείται για την παραγωγή των κορυφαίων προτεινόμενων αντικειμένων, βασισμένων στις υψηλότερες προβλεπόμενες τιμές προτίμησης (Hu et al., 2008).

4.2.1 Ανάλυση αρχείου collaborative.py

```

1  import os
2  import pickle
3  import numpy as np
4  import pandas as pd
5  from scipy.sparse import coo_matrix
6  from implicit.als import AlternatingLeastSquares
7  from sklearn.metrics import ndcg_score
8  from sklearn.metrics.pairwise import cosine_similarity
9
10 # SETTINGS
11 TOP_K = 10
12 EVAL_SAMPLE_USERS = 300
13 EVAL_THRESHOLD = 3.5
14
15 os.environ["OPENBLAS_NUM_THREADS"] = "1" # prevent multithreading issues
16
17 # PATHS
18 train_path = "../data/ml-32m/train.csv"
19 test_path = "../data/ml-32m/test.csv"
20 movies_path = "../data/ml-32m/movies.csv"
21 MODEL_PATH = "../data/ml-32m/als_model_new.pkl"

```

Εικόνα 50: Ρυθμίσεις collaborative filtering μοντέλου

Στο αρχικό τμήμα του collaborative recommender πραγματοποιείται η εισαγωγή των βασικών βιβλιοθηκών και ο ορισμός των παραμέτρων και των διαδρομών που θα χρησιμοποιηθούν σε όλη την υλοποίηση. Η βιβλιοθήκη `os` εισάγεται για τον χειρισμό μεταβλητών περιβάλλοντος και διαδρομών αρχείων, ενώ η `pickle` χρησιμοποιείται για την αποθήκευση και επαναφόρτωση του εκπαιδευμένου μοντέλου ALS από τον δίσκο. Οι βιβλιοθήκες `numpy` και `pandas` καλύπτουν τις αριθμητικές πράξεις και τη διαχείριση των πινάκων δεδομένων αντίστοιχα. Από τη `scipy.sparse` χρησιμοποιείται η `coo_matrix`, η οποία είναι απαραίτητη για την κατασκευή της αραιής μήτρας αλληλεπιδράσεων χρηστών-ταινιών, πάνω στην οποία θα εκπαιδευτεί το collaborative μοντέλο. Παράλληλα, από τη βιβλιοθήκη `implicit.als` εισάγεται η κλάση `AlternatingLeastSquares`, η οποία αποτελεί τον βασικό αλγόριθμο collaborative filtering της παρούσας υλοποίησης. Επιπλέον, χρησιμοποιείται η `ndcg_score` για τον υπολογισμό της μετρικής NDCG κατά την αξιολόγηση, καθώς και η `cosine_similarity`, η οποία αξιοποιείται αργότερα για τον υπολογισμό της ποικιλίας των προτάσεων με βάση τα latent item factors του μοντέλου.

Στη συνέχεια ορίζονται οι βασικές ρυθμίσεις του συστήματος. Η μεταβλητή `TOP_K = 10` καθορίζει ότι το μοντέλο θα επιστρέφει δέκα προτάσεις για κάθε χρήστη, η `EVAL_SAMPLE_USERS = 300` ορίζει το μέγεθος του δείγματος χρηστών που θα χρησιμοποιηθεί στην αξιολόγηση, ενώ η `EVAL_THRESHOLD = 3.5` καθορίζει το κατώφλι πάνω από το οποίο μια ταινία του test set θεωρείται relevant. Αμέσως μετά, με την εντολή `os.environ["OPENBLAS_NUM_THREADS"] = "1"` περιορίζεται η χρήση πολυνηματικής επεξεργασίας από το OpenBLAS σε ένα νήμα, ώστε να αποφευχθούν πιθανά προβλήματα ή αστάθειες κατά την εκτέλεση του αλγορίθμου ALS σε ορισμένα περιβάλλοντα.

Τέλος, ορίζονται οι διαδρομές προς τα βασικά αρχεία του project. Οι μεταβλητές `train_path`, `test_path` και `movies_path` δείχνουν στα αρχεία `train.csv`, `test.csv` και `movies.csv` αντίστοιχα, τα οποία περιέχουν τα δεδομένα εκπαίδευσης, αξιολόγησης και τις βασικές πληροφορίες των ταινιών. Η μεταβλητή `MODEL_PATH` δείχνει στο αρχείο `als_model_new.pkl`, το οποίο χρησιμοποιείται για την αποθήκευση του εκπαιδευμένου μοντέλου ALS, ώστε αυτό να μπορεί να επαναφορτωθεί σε επόμενη εκτέλεση χωρίς να απαιτείται νέα εκπαίδευση από την αρχή. Με αυτόν τον τρόπο ολοκληρώνεται το αρχικό στάδιο προετοιμασίας της collaborative υλοποίησης και τίθεται το πλαίσιο πάνω στο οποίο θα αναπτυχθεί ο υπόλοιπος κώδικας.

```

23 # METRICS HELPERS
24 def precision_at_k(recommended_items, relevant_items, k): 1usage new*
25     hits = sum(1 for item in recommended_items[:k] if item in relevant_items)
26     return hits / k if k > 0 else 0.0
27
28 def recall_at_k(recommended_items, relevant_items, k): 1usage new*
29     if len(relevant_items) == 0:
30         return 0.0
31     hits = sum(1 for item in recommended_items[:k] if item in relevant_items)
32     return hits / len(relevant_items)
33
34 def average_precision_at_k(recommended_items, relevant_items, k): 1usage new*
35     hits = 0
36     precision_sum = 0.0
37
38     for i, item in enumerate(recommended_items[:k], start=1):
39         if item in relevant_items:
40             hits += 1
41             precision_sum += hits / i
42
43     denom = min(len(relevant_items), k)
44     return precision_sum / denom if denom > 0 else 0.0
45
46 def ndcg_at_k(recommended_items, scores, relevant_items, k): 1usage new*
47     y_true = np.array([1 if item in relevant_items else 0 for item in recommended_items[:k]])
48     y_score = np.array([scores[:k]])
49     return ndcg_score(y_true, y_score)
50
51 def hit_rate_at_k(recommended_items, relevant_items, k): 1usage new*
52     return 1.0 if any(item in relevant_items for item in recommended_items[:k]) else 0.0

```

Εικόνα 51: Μετρικές αξιολόγησης collaborative filtering

Στο επόμενο τμήμα του collaborative recommender ορίζονται οι βασικές συναρτήσεις αξιολόγησης, οι οποίες χρησιμοποιούνται για τη μέτρηση της ποιότητας των προτάσεων του μοντέλου. Οι συναρτήσεις αυτές είναι ίδιες σε λογική με εκείνες που χρησιμοποιήθηκαν και στο content-based σύστημα, ώστε η αξιολόγηση των δύο προσεγγίσεων να πραγματοποιείται με κοινό και άμεσα συγκρίσιμο τρόπο. Η συνάρτηση `precision_at_k` υπολογίζει το ποσοστό των προτεινόμενων ταινιών στις πρώτες k θέσεις που είναι πράγματι relevant για τον χρήστη. Για τον σκοπό αυτό, ο κώδικας μετρά πόσα από τα πρώτα k αντικείμενα της λίστας `recommended_items` ανήκουν στο σύνολο `relevant_items` και στη συνέχεια διαιρεί το πλήθος αυτό με το k . Έτσι, η μετρική αυτή δείχνει πόσο “καθαρή” είναι η λίστα προτάσεων, δηλαδή πόσες από τις πρώτες θέσεις καταλαμβάνονται από σωστές επιλογές.

Η συνάρτηση `recall_at_k` μετρά πόσο μεγάλο μέρος των relevant αντικειμένων του χρήστη καταφέρνει να ανακτήσει το σύστημα μέσα στις πρώτες k θέσεις. Αν το σύνολο `relevant_items` είναι κενό, επιστρέφεται τιμή 0.0, καθώς δεν υπάρχουν σχετικά αντικείμενα

προς ανάκτηση. Διαφορετικά, μετριοούνται και εδώ οι επιτυχίες μέσα στο top-k και το πλήθος αυτό διαιρείται με το συνολικό πλήθος των relevant ταινιών. Με αυτόν τον τρόπο, η Recall δεν δείχνει πόσο σωστές είναι οι προτάσεις, αλλά πόσο καλά καλύπτει το σύστημα το σύνολο των ταινιών που θεωρούνται σχετικές για τον χρήστη.

Η συνάρτηση `average_precision_at_k` προχωρά ένα βήμα παραπέρα, καθώς δεν εξετάζει μόνο το αν βρέθηκαν relevant ταινίες, αλλά και τη θέση στην οποία εμφανίστηκαν. Ο κώδικας διατρέχει τις πρώτες k προτάσεις με τη βοήθεια της `enumerate`, κρατώντας ταυτόχρονα τη θέση και το αντικείμενο. Κάθε φορά που εντοπίζεται relevant ταινία, αυξάνεται ο μετρητής hits και προστίθεται στο `precision_sum` η ακρίβεια που έχει επιτευχθεί μέχρι εκείνο το σημείο της λίστας. Στο τέλος, το άθροισμα αυτό κανονικοποιείται με βάση το μικρότερο μεταξύ του πλήθους των relevant αντικειμένων και του k. Έτσι, η συγκεκριμένη μετρική επιβραβεύει τις relevant ταινίες που εμφανίζονται νωρίτερα στη λίστα και αποτυπώνει πιο ουσιαστικά την ποιότητα της κατάταξης.

Η συνάρτηση `ndcg_at_k` χρησιμοποιείται για τον υπολογισμό της μετρικής NDCG@K, η οποία αξιολογεί επίσης την ποιότητα της διάταξης των προτάσεων, αλλά με ακόμη μεγαλύτερη έμφαση στις πρώτες θέσεις. Για τον σκοπό αυτό, ο κώδικας δημιουργεί πρώτα το διάνυσμα `y_true`, στο οποίο κάθε θέση λαμβάνει τιμή 1 αν η αντίστοιχη προτεινόμενη ταινία ανήκει στα `relevant_items` και 0 διαφορετικά. Παράλληλα δημιουργείται και το διάνυσμα `y_score`, το οποίο περιέχει τα scores των πρώτων k προτάσεων. Τα δύο αυτά διανύσματα δίνονται στη συνάρτηση `ndcg_score`, η οποία υπολογίζει την τελική τιμή της NDCG. Με αυτόν τον τρόπο η μετρική εξετάζει όχι μόνο αν υπάρχουν relevant αντικείμενα, αλλά και κατά πόσο αυτά βρίσκονται στις πρώτες θέσεις της κατάταξης, όπου είναι περισσότερο χρήσιμα για τον χρήστη.

Τελικώς, η συνάρτηση `hit_rate_at_k` υλοποιεί μια πιο απλή αλλά πολύ πρακτική μετρική. Ελέγχει αν υπάρχει έστω μία relevant ταινία μέσα στις πρώτες k προτάσεις και επιστρέφει 1.0 αν υπάρχει τουλάχιστον μία επιτυχία ή 0.0 διαφορετικά. Η μετρική αυτή δείχνει ουσιαστικά αν το σύστημα κατάφερε να προσφέρει τουλάχιστον μία χρήσιμη πρόταση στον χρήστη μέσα στο top-k. Συνολικά, το σύνολο αυτών των συναρτήσεων διαμορφώνει το βασικό πλαίσιο αξιολόγησης του collaborative recommender και επιτρέπει τη μελέτη της απόδοσής του τόσο ως προς τη συνάφεια των προτάσεων όσο και ως προς την ποιότητα της κατάταξης.

```

54 def intra_list_diversity_cf(recommended_items): 1 usage new *
55     """
56     Diversity based on ALS latent item factors.
57     1 - average cosine similarity among recommended item vectors.
58     """
59     item_indices = [item_map[item] for item in recommended_items if item in item_map]
60
61     if len(item_indices) < 2:
62         return 0.0
63
64     item_vectors = als.item_factors[item_indices]
65     sim_matrix = cosine_similarity(item_vectors)
66
67     n = len(item_indices)
68     pairwise_sims = []
69
70     for i in range(n):
71         for j in range(i + 1, n):
72             pairwise_sims.append(sim_matrix[i, j])
73
74     if not pairwise_sims:
75         return 0.0
76
77     return 1.0 - np.mean(pairwise_sims)

```

Εικόνα 52: Ποικιλομορφία συστάσεων collaborative filtering

Η συνάρτηση `intra_list_diversity_cf` χρησιμοποιείται για τον υπολογισμό της ποικιλίας της τελικής λίστας προτάσεων στο collaborative σύστημα. Όπως και στην περίπτωση του content-based μοντέλου, στόχος της δεν είναι να μετρήσει αν οι προτεινόμενες ταινίες είναι σωστές σε σχέση με τις προτιμήσεις του χρήστη, αλλά κατά πόσο διαφέρουν μεταξύ τους μέσα στην ίδια λίστα. Η βασική διαφορά εδώ είναι ότι η ποικιλία δεν υπολογίζεται πάνω σε TF-IDF διανύσματα περιεχομένου, αλλά πάνω στα latent item factors που έχει μάθει το μοντέλο ALS. Με άλλα λόγια, η ομοιότητα ανάμεσα στις προτεινόμενες ταινίες εξετάζεται στον λανθάνοντα χώρο αναπαράστασης του collaborative filtering.

Αρχικά, ο κώδικας μετατρέπει τα `movieId` των προτεινόμενων ταινιών στα αντίστοιχα εσωτερικά `indices` μέσω του λεξικού `item_map`. Με αυτόν τον τρόπο, κάθε ταινία αντιστοιχίζεται στη σωστή θέση μέσα στους latent παράγοντες του μοντέλου. Αν μετά από αυτή τη διαδικασία υπάρχουν λιγότερες από δύο έγκυρες ταινίες, η συνάρτηση επιστρέφει 0.0, καθώς δεν μπορεί να υπολογιστεί diversity χωρίς τουλάχιστον δύο αντικείμενα προς σύγκριση. Στη συνέχεια, από το `als.item_factors` ανακτώνται τα διανύσματα των συγκεκριμένων ταινιών και πάνω σε αυτά υπολογίζεται ο πίνακας ομοιότητας `sim_matrix` με χρήση της

cosine_similarity. Ο πίνακας αυτός δείχνει πόσο κοντά βρίσκονται μεταξύ τους οι προτεινόμενες ταινίες στον λανθάνοντα χώρο που έχει μάθει το μοντέλο ALS.

Στο επόμενο στάδιο, ο κώδικας εξετάζει όλα τα μοναδικά ζεύγη ταινιών της λίστας και αποθηκεύει τις επιμέρους τιμές ομοιότητας στη λίστα pairwise_sims. Η εσωτερική επανάληψη ξεκινά από το $i + 1$, ώστε να αποφεύγεται τόσο η σύγκριση μιας ταινίας με τον εαυτό της όσο και ο διπλός υπολογισμός του ίδιου ζεύγους με αντίστροφη σειρά. Αν η λίστα pairwise_sims παραμείνει κενή, η συνάρτηση επιστρέφει 0.0 για λόγους ασφαλείας. Τελικά, η diversity υπολογίζεται ως $1.0 - \text{μέσος όρος pairwise similarity}$. Αυτό σημαίνει ότι όσο πιο όμοιες είναι μεταξύ τους οι προτεινόμενες ταινίες στον λανθάνοντα χώρο του ALS, τόσο μικρότερη θα είναι η τελική τιμή της diversity. Αντίθετα, όσο περισσότερο διαφοροποιούνται, τόσο μεγαλύτερη θα είναι η τιμή της. Έτσι, η συνάρτηση αποτυπώνει τον βαθμό εσωτερικής ποικιλίας της λίστας προτάσεων του collaborative recommender, χρησιμοποιώντας αυτή τη φορά τη learned collaborative αναπαράσταση των ταινιών αντί για το περιεχόμενό τους.

```

79 # LOAD DATA
80 train_df = pd.read_csv(train_path)
81 test_df = pd.read_csv(test_path)
82 movies_df = pd.read_csv(movies_path)
83
84 for col in ["title", "genres"]:
85     if col in train_df.columns:
86         train_df = train_df.drop(columns=[col])
87     if col in test_df.columns:
88         test_df = test_df.drop(columns=[col])
89
90 print(f"Movies raw: {movies_df.shape}")
91 print(f"Train raw: {train_df.shape}")
92 print(f"Test raw: {test_df.shape}")
93
94 # =====
95 # CANDIDATE CATALOG = TRAIN ITEMS ONLY
96 # =====
97 train_item_ids = set(train_df["movieId"].unique())
98 test_item_ids = set(test_df["movieId"].unique())
99
100 print(f"Unique movies in original movies.csv: {movies_df['movieId'].nunique()}")
101 print(f"Unique movies in TRAIN: {len(train_item_ids)}")
102 print(f"Unique movies in TEST: {len(test_item_ids)}")
103
104 movies_df = movies_df[movies_df["movieId"].isin(train_item_ids)].copy()
105
106 print(f"Candidate movies for recommendation: {movies_df['movieId'].nunique()}")

```

Εικόνα 53: Δημιουργία καταλόγου υποψήφιων ταινιών στο collaborative filtering

Στο επόμενο στάδιο του collaborative recommender πραγματοποιείται η φόρτωση των βασικών δεδομένων που θα χρησιμοποιηθούν για την εκπαίδευση και την αξιολόγηση του μοντέλου. Συγκεκριμένα, τα αρχεία train.csv, test.csv και movies.csv διαβάζονται σε μορφή DataFrame μέσω της pd.read_csv και αποθηκεύονται στις μεταβλητές train_df, test_df και movies_df αντίστοιχα. Αμέσως μετά, εφαρμόζεται ένας καθαρισμός στα train_df και test_df, όπου αφαιρούνται οι στήλες title και genres εφόσον υπάρχουν. Η ενέργεια αυτή γίνεται μόνο στη μνήμη και δεν μεταβάλλει τα αρχικά αρχεία, ενώ στόχος της είναι να διατηρηθούν τα train και test σύνολα όσο το δυνατόν πιο καθαρά και συγκρίσιμα με τη λογική που ακολουθήθηκε και στο content-based script, αφήνοντας τον πίνακα movies_df ως την κύρια πηγή μεταδεδομένων για τις ταινίες.

Εν συνεχεία, ο κώδικας εκτυπώνει τις διαστάσεις των τριών DataFrames ώστε να επιβεβαιωθεί το αρχικό μέγεθος των δεδομένων που εισέρχονται στο σύστημα. Ακολουθεί ο ορισμός του candidate catalog, δηλαδή του συνόλου των ταινιών από το οποίο το collaborative μοντέλο θα μπορεί τελικά να παράγει προτάσεις. Για τον σκοπό αυτό, εξάγονται πρώτα τα μοναδικά movieId που εμφανίζονται στο training set και στο test set, και αποθηκεύονται στα σύνολα train_item_ids και test_item_ids. Έπειτα, ο κώδικας εμφανίζει το πλήθος των μοναδικών ταινιών στο αρχικό movies.csv, καθώς και το πλήθος των μοναδικών ταινιών που εμφανίζονται στο train και στο test, ώστε να γίνει σαφές ποιο μέρος του αρχικού καταλόγου συμμετέχει πραγματικά στα δύο splits. Όπως και στην περίπτωση του content-based recommender, εδώ πρέπει να διευκρινιστεί ότι το movies.csv αποτελεί τον πλήρη αρχικό κατάλογο ταινιών, ενώ τα train_df και test_df είναι split αξιολογήσεων και όχι split του ίδιου του καταλόγου ταινιών. Επομένως, οι ταινίες που συμμετέχουν σε κάθε split προκύπτουν έμμεσα από τα movieId που εμφανίζονται στα αντίστοιχα σύνολα αξιολογήσεων.

Συνοψίζοντας, ο πίνακας movies_df φιλτράρεται έτσι ώστε να διατηρηθούν μόνο οι ταινίες των οποίων το movieId εμφανίζεται στο training set. Με άλλα λόγια, το candidate catalog του collaborative συστήματος περιορίζεται αποκλειστικά στα items του train. Η επιλογή αυτή είναι μεθοδολογικά συνεπής, καθώς το μοντέλο ALS εκπαιδεύεται πάνω στις αλληλεπιδράσεις του training set και συνεπώς οι προτάσεις του πρέπει να παράγονται μόνο για ταινίες που ανήκουν σε αυτόν τον χώρο μάθησης. Η τελική εκτύπωση του πλήθους των ταινιών που παραμένουν μετά το φιλτράρισμα δίνει το μέγεθος του τελικού candidate catalog, δηλαδή του συνόλου των αντικειμένων από το οποίο το collaborative recommender θα αντλεί τις συστάσεις του.

```

108     # MAP IDs TO INDICES
109     user_ids = train_df["userId"].unique()
110     item_ids = train_df["movieId"].unique()
111
112     user_map = {uid: i for i, uid in enumerate(user_ids)}
113     item_map = {iid: i for i, iid in enumerate(item_ids)}
114
115     user_map_rev = {v: k for k, v in user_map.items()}
116     item_map_rev = {v: k for k, v in item_map.items()}
117
118     # Align train/test to train catalog
119     train_df = train_df[train_df["movieId"].isin(item_map)].copy()
120     test_df = test_df[test_df["movieId"].isin(item_map)].copy()
121
122     print(f"Train after alignment: {train_df.shape}")
123     print(f"Test after alignment: {test_df.shape}")
124     print(f"Final candidate movie count: {len(item_map)}")
125
126     # CREATE INTERACTION MATRIX
127     rows = train_df["userId"].map(user_map)
128     cols = train_df["movieId"].map(item_map)
129     data = train_df["rating"]
130
131     interaction_matrix = coo_matrix(
132         arg1=(data, (rows, cols)),
133         shape=(len(user_ids), len(item_ids))
134     )
135
136     print(f"Interaction matrix shape: {interaction_matrix.shape}")

```

Εικόνα 54: Δημιουργία πίνακα αλληλεπιδράσεων

Κατόπιν, πραγματοποιείται η αντιστοίχιση των πραγματικών αναγνωριστικών χρηστών και ταινιών σε εσωτερικούς ακέραιους δείκτες, ώστε τα δεδομένα να αποκτήσουν μορφή κατάλληλη για την κατασκευή της μήτρας αλληλεπιδράσεων. Αρχικά, από το `train_df` εξάγονται τα μοναδικά `userId` και `movieId`, τα οποία αποθηκεύονται στις μεταβλητές `user_ids` και `item_ids`. Στη συνέχεια δημιουργούνται τα λεξικά `user_map` και `item_map`, στα οποία κάθε χρήστης και κάθε ταινία αντιστοιχίζεται σε έναν ακέραιο δείκτη. Η διαδικασία αυτή είναι

απαραίτητη, επειδή το μοντέλο ALS και οι sparse matrix δομές δεν δουλεύουν άμεσα πάνω στα αρχικά αναγνωριστικά του dataset, αλλά απαιτούν συνεχόμενη αριθμητική αρίθμηση γραμμών και στηλών. Παράλληλα, δημιουργούνται και τα αντίστροφα λεξικά `user_map_rev` και `item_map_rev`, ώστε να είναι δυνατή αργότερα η επιστροφή από τους εσωτερικούς δείκτες στα πραγματικά `userId` και `movieId`, κάτι που είναι ιδιαίτερα χρήσιμο στο στάδιο παραγωγής των τελικών συστάσεων.

Αμέσως μετά πραγματοποιείται η ευθυγράμμιση των `train_df` και `test_df` με το τελικό candidate catalog του training set. Συγκεκριμένα, διατηρούνται μόνο οι εγγραφές των οποίων το `movieId` περιλαμβάνεται στο `item_map`, δηλαδή μόνο όσες αφορούν ταινίες που έχουν παραμείνει στο train-based recommendation space. Με αυτόν τον τρόπο εξασφαλίζεται ότι τόσο το training όσο και το test σύνολο περιέχουν μόνο αντικείμενα που είναι γνωστά στο μοντέλο και μπορούν να αντιστοιχιστούν σωστά στη μήτρα αλληλεπιδράσεων. Οι σχετικές εκτυπώσεις που ακολουθούν επιτρέπουν να επιβεβαιωθεί το μέγεθος των `train_df` και `test_df` μετά την ευθυγράμμιση, καθώς και το τελικό πλήθος των υποψήφιων ταινιών.

Παράλληλα, κατασκευάζεται η μήτρα αλληλεπιδράσεων χρηστών-ταινιών. Για τον σκοπό αυτό, οι στήλες `userId` και `movieId` του `train_df` μετατρέπονται στους αντίστοιχους εσωτερικούς δείκτες μέσω των λεξικών `user_map` και `item_map` και αποθηκεύονται στις μεταβλητές `rows` και `cols`, ενώ η στήλη `rating` αποθηκεύεται στη μεταβλητή `data`. Έτσι, κάθε εγγραφή του training set παίρνει τη μορφή (γραμμή χρήστη, στήλη ταινίας, τιμή `rating`). Με βάση αυτά τα τρία σύνολα πληροφοριών δημιουργείται η `interaction_matrix` με χρήση της `coo_matrix`, όπου το σχήμα της μήτρας ορίζεται από το πλήθος των μοναδικών χρηστών και των μοναδικών ταινιών του train. Το αποτέλεσμα είναι μια sparse μήτρα, στην οποία οι γραμμές αντιστοιχούν σε χρήστες, οι στήλες σε ταινίες και τα μη μηδενικά κελιά στις καταγεγραμμένες αξιολογήσεις. Η επιλογή sparse αναπαράστασης είναι ιδιαίτερα σημαντική, καθώς σε προβλήματα recommendation ο πίνακας χρηστών-ταινιών είναι εξαιρετικά αραιός και η αποθήκευσή του σε dense μορφή θα ήταν υπολογιστικά αναποτελεσματική. Τέλος, με την εκτύπωση του σχήματος της `interaction_matrix` επιβεβαιώνεται η τελική διάσταση του πίνακα πάνω στον οποίο θα εκπαιδευτεί το collaborative μοντέλο ALS.

```
136 # TRAIN OR LOAD ALS
137 if os.path.exists(MODEL_PATH):
138     with open(MODEL_PATH, "rb") as f:
139         als = pickle.load(f)
140         print("ALS model loaded from disk")
141 else:
142     als = AlternatingLeastSquares(
143         factors=50,
144         regularization=0.1,
145         iterations=20,
146         random_state=42
147     )
148     als.fit(interaction_matrix.tocsr())
149     print("ALS model trained")
150
151     with open(MODEL_PATH, "wb") as f:
152         pickle.dump(als, f)
153     print("ALS model saved to disk")
```

Εικόνα 55: Εκπαίδευση και αποθήκευση ALS μοντέλου

Προχωρώντας, ο κώδικας ελέγχει αν υπάρχει ήδη αποθηκευμένο στον δίσκο ένα εκπαιδευμένο μοντέλο ALS. Η εντολή `if os.path.exists(MODEL_PATH)`: χρησιμοποιείται για να διαπιστωθεί αν το αρχείο που έχει οριστεί στη διαδρομή `MODEL_PATH` είναι ήδη διαθέσιμο. Αν το αρχείο υπάρχει, τότε ανοίγεται σε λειτουργία ανάγνωσης δυαδικών δεδομένων ("rb") και με τη χρήση της `pickle.load(f)` το μοντέλο φορτώνεται απευθείας στη μεταβλητή `als`. Στην περίπτωση αυτή, ο κώδικας εμφανίζει το μήνυμα "ALS model loaded from disk", δηλώνοντας ότι δεν χρειάστηκε νέα εκπαίδευση. Η επιλογή αυτή είναι ιδιαίτερα χρήσιμη σε μεγάλα datasets, καθώς η εκ νέου εκπαίδευση του μοντέλου μπορεί να απαιτεί σημαντικό χρόνο και υπολογιστικούς πόρους. Επομένως, η αποθήκευση και επαναφόρτωση του μοντέλου επιτρέπει πιο αποδοτική ανάπτυξη και επαναληπτικό πειραματισμό.

Αν το μοντέλο δεν υπάρχει ήδη αποθηκευμένο, τότε ο κώδικας περνά στο στάδιο της εκπαίδευσης. Σε αυτή την περίπτωση δημιουργείται ένα νέο αντικείμενο

AlternatingLeastSquares, στο οποίο ορίζονται οι βασικές υπερπαράμετροι του αλγορίθμου: `factors=50`, `regularization=0.1`, `iterations=20` και `random_state=42`. Η παράμετρος `factors` καθορίζει το μέγεθος του λανθάνοντος χώρου, δηλαδή πόσες διαστάσεις θα χρησιμοποιηθούν για την αναπαράσταση χρηστών και ταινιών. Η `regularization` λειτουργεί ως μηχανισμός ελέγχου υπερπροσαρμογής, περιορίζοντας την υπερβολική προσαρμογή του μοντέλου στα δεδομένα εκπαίδευσης, ενώ η `iterations` ορίζει πόσες επαναλήψεις θα εκτελέσει ο αλγόριθμος κατά τη διαδικασία βελτιστοποίησης. Η χρήση της `random_state=42` διασφαλίζει αναπαραγωγιμότητα στα αποτελέσματα της εκπαίδευσης. Στη συνέχεια, το μοντέλο εκπαιδεύεται πάνω στη μήτρα αλληλεπιδράσεων μέσω της `als.fit(interaction_matrix.tocsr())`. Η μετατροπή της μήτρας σε μορφή CSR γίνεται ώστε η sparse αναπαράσταση να είναι κατάλληλη και αποδοτική για τη διαδικασία εκπαίδευσης του ALS.

Μετά την ολοκλήρωση της εκπαίδευσης, εμφανίζεται το μήνυμα "ALS model trained" και ακολουθεί η αποθήκευση του μοντέλου στον δίσκο. Για τον σκοπό αυτό, το αρχείο που αντιστοιχεί στο `MODEL_PATH` ανοίγεται σε λειτουργία εγγραφής δυαδικών δεδομένων ("wb") και το εκπαιδευμένο αντικείμενο `als` αποθηκεύεται με χρήση της `pickle.dump(als, f)`. Τέλος, εμφανίζεται το μήνυμα "ALS model saved to disk", επιβεβαιώνοντας ότι το μοντέλο είναι πλέον διαθέσιμο για επόμενες εκτελέσεις χωρίς νέα εκπαίδευση. Συνολικά, το τμήμα αυτό του κώδικα διασφαλίζει ότι η collaborative υλοποίηση μπορεί να λειτουργεί αποδοτικά, επιλέγοντας είτε την άμεση επαναχρησιμοποίηση ενός ήδη εκπαιδευμένου μοντέλου είτε την εκ νέου εκπαίδευσή του όταν αυτό απαιτείται.

```

155 # TOP-N RECOMMENDATIONS
156 def recommend_top_n(model, train_matrix, user_id, item_df, n=10): 6 usages  ⤴ PanosKarak *
157     if user_id not in user_map:
158         return pd.DataFrame(columns=["movieId", "title", "score"])
159
160     user_idx = user_map[user_id]
161
162     recs, scores = model.recommend(
163         userid=user_idx,
164         user_items=train_matrix.tocsr()[user_idx],
165         N=n
166     )
167
168     recs_df = pd.DataFrame({
169         "item_idx": recs.astype(int),
170         "score": scores
171     })
172
173     recs_df["movieId"] = recs_df["item_idx"].map(item_map_rev)
174     recs_df = recs_df.merge(item_df, on="movieId", how="left")
175
176     return recs_df[["movieId", "title", "score"]].sort_values("score", ascending=False).reset_index(drop=True)

```

Εικόνα 56: Παραγωγή Top-N συστάσεων με ALS

Η συνάρτηση `recommend_top_n` υλοποιεί το στάδιο παραγωγής προτάσεων για έναν συγκεκριμένο χρήστη στο collaborative σύστημα. Αρχικά, ο κώδικας ελέγχει αν το `user_id` υπάρχει στο λεξικό `user_map`. Αν ο χρήστης δεν υπάρχει, επιστρέφεται ένα κενό `DataFrame` με τις στήλες `movieId`, `title` και `score`, καθώς δεν είναι δυνατό να παραχθούν προτάσεις για χρήστη που δεν ανήκει στον χώρο εκπαίδευσης του μοντέλου. Αν ο χρήστης υπάρχει, το πραγματικό του αναγνωριστικό μετατρέπεται στον αντίστοιχο εσωτερικό δείκτη μέσω της μεταβλητής `user_idx`, ώστε να μπορεί να χρησιμοποιηθεί από το μοντέλο ALS.

Έπειτα, καλείται η μέθοδος `model.recommend`, η οποία αποτελεί τον βασικό μηχανισμό παραγωγής προτάσεων του ALS. Ως είσοδος δίνονται ο δείκτης του χρήστη (`userid=user_idx`), οι υπάρχουσες αλληλεπιδράσεις του χρήστη από τη μήτρα εκπαίδευσης (`user_items=train_matrix.tocsr()[user_idx]`) και το πλήθος των προτάσεων που ζητούνται (`N=n`). Η μέθοδος επιστρέφει δύο πίνακες: τον πίνακα `recs`, που περιέχει τους εσωτερικούς δείκτες των προτεινόμενων ταινιών, και τον πίνακα `scores`, που περιέχει το αντίστοιχο collaborative score κάθε πρότασης. Η χρήση του `user_items` είναι σημαντική, καθώς επιτρέπει στο μοντέλο να γνωρίζει ποιες ταινίες έχει ήδη δει ο χρήστης στο training set και έτσι να μην τις επιστρέφει ξανά ως νέες προτάσεις.

Ακολούθως, τα αποτελέσματα μετατρέπονται σε `DataFrame` με τις στήλες `item_idx` και `score`, ώστε να είναι πιο εύκολα διαχειρίσιμα. Στη συνέχεια, οι εσωτερικοί δείκτες των ταινιών μετατρέπονται ξανά στα πραγματικά `movieId` μέσω του λεξικού `item_map_rev`, και ακολουθεί συγχώνευση με τον πίνακα `item_df` ώστε να προστεθεί ο τίτλος κάθε ταινίας. Με αυτόν τον τρόπο, οι τελικές προτάσεις αποκτούν αναγνώσιμη μορφή και δεν περιορίζονται μόνο σε εσωτερικούς αριθμητικούς δείκτες. Τέλος, η συνάρτηση επιστρέφει μόνο τις στήλες `movieId`, `title` και `score`, ταξινομημένες κατά φθίνουσα σειρά `score` και με ανανέωση των δεικτών. Έτσι, το αποτέλεσμα είναι μια καθαρή και πλήρως μορφοποιημένη λίστα με τις top-N collaborative προτάσεις για τον συγκεκριμένο χρήστη.

```
178 # EVALUATION
179 def evaluate_model(sample_size=300, top_k=10, eval_threshold=3.5):
180     precision_scores = []
181     recall_scores = []
182     map_scores = []
183     ndcg_scores_list = []
184     hitrate_scores = []
185     diversity_scores = []
186
187     unique_test_users = test_df["userId"].drop_duplicates()
188     sample_users = unique_test_users.sample(
189         min(sample_size, len(unique_test_users)),
190         random_state=42
191     )
192
193     evaluated_users = 0
194
195     for uid in sample_users:
196         if uid not in user_map:
197             continue
198
199         user_test = test_df[test_df["userId"] == uid].copy()
200
201         # Relevant items μόνο μέσα στο train-known catalog
202         relevant_items = set(
203             user_test[user_test["rating"] >= eval_threshold]["movieId"].unique()
204         )
205
206         if len(relevant_items) == 0:
207             continue
```

Εικόνα 57: Αξιολόγηση ALS collaborative filtering μοντέλου

Η συνάρτηση `evaluate_model` ορίζει το πλαίσιο αξιολόγησης του collaborative recommender και συγκεντρώνει τις βασικές παραμέτρους που απαιτούνται για τον υπολογισμό των μετρικών απόδοσης. Δέχεται ως παραμέτρους το μέγεθος του δείγματος χρηστών (`sample_size`), το πλήθος των προτάσεων που θα εξεταστούν (`top_k`) και το κατώφλι με το οποίο μια ταινία θεωρείται relevant στο test set (`eval_threshold`). Στην αρχή της συνάρτησης δημιουργούνται έξι ξεχωριστές λίστες, `precision_scores`, `recall_scores`, `map_scores`, `ndcg_scores_list`, `hitrate_scores` και `diversity_scores`, μέσα στις οποίες θα αποθηκεύονται οι επιμέρους τιμές των αντίστοιχων μετρικών για κάθε χρήστη που θα αξιολογηθεί. Με αυτόν τον τρόπο, η αξιολόγηση οργανώνεται εξαρχής ώστε να μπορεί στο τέλος να υπολογιστεί ο μέσος όρος κάθε μετρικής στο σύνολο του δείγματος.

Στο επόμενο βήμα, από το `test_df` εξάγονται οι μοναδικοί χρήστες μέσω της `drop_duplicates()`, ώστε κάθε χρήστης να εμφανίζεται μόνο μία φορά στη διαδικασία αξιολόγησης. Από αυτό το σύνολο επιλέγεται έπειτα τυχαίο δείγμα χρηστών με χρήση της `sample()`. Το μέγεθος του δείγματος καθορίζεται από το μικρότερο μεταξύ της τιμής `sample_size` και του πλήθους των διαθέσιμων χρηστών, ώστε να αποφεύγεται σφάλμα σε περίπτωση που ζητηθεί μεγαλύτερο δείγμα από το πραγματικά διαθέσιμο. Η επιλογή `random_state=42` εξασφαλίζει ότι η ίδια δειγματοληψία μπορεί να αναπαραχθεί σε επόμενες εκτελέσεις του πειράματος, γεγονός σημαντικό για τη συνέπεια και τη συγκρισιμότητα των αποτελεσμάτων. Αμέσως μετά, η μεταβλητή `evaluated_users` αρχικοποιείται στο μηδέν και θα χρησιμοποιηθεί αργότερα για να καταγράψει πόσοι χρήστες αξιολογήθηκαν τελικά, αφού ενδέχεται κάποιοι να αποκλειστούν στα επόμενα βήματα.

Το κύριο μέρος του πρώτου αυτού τμήματος της συνάρτησης αφορά τον εντοπισμό των χρηστών που είναι πράγματι κατάλληλοι για αξιολόγηση. Για κάθε χρήστη του δείγματος, ο κώδικας ελέγχει πρώτα αν ο χρήστης υπάρχει στο `user_map`. Ο έλεγχος αυτός είναι απαραίτητος, επειδή το collaborative μοντέλο ALS μπορεί να παράγει προτάσεις μόνο για χρήστες που ανήκουν στον χώρο εκπαίδευσης του training set και άρα έχουν αντιστοίχιση σε εσωτερικό δείκτη. Αν ο χρήστης δεν υπάρχει στο `user_map`, παραλείπεται άμεσα. Στη συνέχεια, ανακτάται το υποσύνολο του `test_df` που αντιστοιχεί στον συγκεκριμένο χρήστη και αποθηκεύεται στη μεταβλητή `user_test`. Πάνω σε αυτό το υποσύνολο ορίζονται τα `relevant_items`, δηλαδή οι ταινίες του χρήστη στο test set που έχουν βαθμολογία μεγαλύτερη ή ίση από το `eval_threshold`. Με άλλα λόγια, το σύστημα θεωρεί relevant μόνο εκείνα τα αντικείμενα που ο χρήστης αξιολόγησε αρκετά θετικά στο test set. Τέλος, αν το σύνολο αυτό είναι κενό, ο χρήστης παραλείπεται επίσης με την εντολή `continue`, καθώς δεν υπάρχει νόημα να υπολογιστούν μετρικές αξιολόγησης για χρήστη που δεν διαθέτει κανένα relevant αντικείμενο. Έτσι, μέχρι αυτό το σημείο η συνάρτηση έχει ολοκληρώσει την επιλογή του δείγματος και τον καθορισμό των χρηστών που μπορούν πράγματι να συμμετάσχουν ουσιαστικά στην αξιολόγηση του collaborative recommender.

```

209     recs_df = recommend_top_n(als, interaction_matrix, uid, movies_df, n=top_k)
210     if recs_df.empty:
211         continue
212
213     recommended_items = recs_df["movieId"].tolist()
214     recommended_scores = recs_df["score"].tolist()
215
216     precision_scores.append(precision_at_k(recommended_items, relevant_items, top_k))
217     recall_scores.append(recall_at_k(recommended_items, relevant_items, top_k))
218     map_scores.append(average_precision_at_k(recommended_items, relevant_items, top_k))
219     ndcg_scores_list.append(ndcg_at_k(recommended_items, recommended_scores, relevant_items, top_k))
220     hitrate_scores.append(hit_rate_at_k(recommended_items, relevant_items, top_k))
221     diversity_scores.append(intra_list_diversity_cf(recommended_items))
222
223     evaluated_users += 1
224
225     return {
226         "Users evaluated": evaluated_users,
227         f"Precision@{top_k}": np.mean(precision_scores) if precision_scores else 0.0,
228         f"Recall@{top_k}": np.mean(recall_scores) if recall_scores else 0.0,
229         f"MAP@{top_k}": np.mean(map_scores) if map_scores else 0.0,
230         f"NDCG@{top_k}": np.mean(ndcg_scores_list) if ndcg_scores_list else 0.0,
231         f"HitRate@{top_k}": np.mean(hitrate_scores) if hitrate_scores else 0.0,
232         "Diversity": np.mean(diversity_scores) if diversity_scores else 0.0
233     }

```

Εικόνα 58: Υπολογισμός μετρικών αξιολόγησης ALS

Στο επόμενο στάδιο της συνάρτησης πραγματοποιείται ο υπολογισμός των προτάσεων και των μετρικών αξιολόγησης για κάθε χρήστη που έχει κριθεί κατάλληλος. Αρχικά, καλείται η `recommend_top_n(als, interaction_matrix, uid, movies_df, n=top_k)`, η οποία παράγει τις top-k collaborative προτάσεις του μοντέλου ALS για τον συγκεκριμένο χρήστη. Αν η συνάρτηση επιστρέψει κενό DataFrame, ο χρήστης παραλείπεται, καθώς δεν υπάρχουν διαθέσιμες προτάσεις για να αξιολογηθούν. Όταν όμως υπάρχουν έγκυρες προτάσεις, ο κώδικας εξάγει από το `recs_df` δύο βασικές λίστες: τη λίστα `recommended_items`, η οποία περιέχει τα `movieId` των προτεινόμενων ταινιών, και τη λίστα `recommended_scores`, η οποία περιέχει τα αντίστοιχα collaborative scores. Οι δύο αυτές λίστες αποτελούν τη βάση πάνω στην οποία θα εφαρμοστούν οι συναρτήσεις αξιολόγησης.

Στη συνέχεια, για κάθε χρήστη υπολογίζονται οι επιμέρους τιμές των μετρικών και αποθηκεύονται στις αντίστοιχες λίστες. Η `precision_at_k` μετρά το ποσοστό των top-k προτάσεων που είναι relevant, η `recall_at_k` δείχνει τι ποσοστό των relevant αντικειμένων του χρήστη ανακτήθηκε μέσα στις πρώτες k θέσεις, ενώ η `average_precision_at_k` λαμβάνει υπόψη και τη σειρά εμφάνισης των relevant ταινιών μέσα στην κατάταξη. Παράλληλα, η `ndcg_at_k` αξιολογεί το κατά πόσο οι σχετικές προτάσεις τοποθετούνται ψηλά στη λίστα, η `hit_rate_at_k` ελέγχει αν υπάρχει τουλάχιστον μία σχετική πρόταση στο top-k, και η `intra_list_diversity_cf`

αποτυπώνει τον βαθμό εσωτερικής ποικιλίας της λίστας προτάσεων με βάση τους latent παράγοντες του ALS. Κάθε τιμή προστίθεται στη λίστα που αντιστοιχεί στη συγκεκριμένη μετρική και, μόλις ολοκληρωθεί επιτυχώς η αξιολόγηση του χρήστη, αυξάνεται κατά ένα η μεταβλητή `evaluated_users`.

Καταληκτικά, η συνάρτηση επιστρέφει ένα λεξικό με τα συνολικά αποτελέσματα της αξιολόγησης. Το πεδίο "Users evaluated" καταγράφει πόσοι χρήστες συμμετείχαν τελικά στην τελική αποτίμηση, ενώ για κάθε μετρική υπολογίζεται ο μέσος όρος των τιμών που συγκεντρώθηκαν στις αντίστοιχες λίστες. Αν κάποια λίστα είναι κενή, επιστρέφεται η τιμή 0.0, ώστε η συνάρτηση να παραμένει ασφαλής και να μην αποτυγχάνει σε ακραίες περιπτώσεις. Με την προσέγγιση αυτή, η `evaluate_model` δεν επιστρέφει αναλυτικά αποτελέσματα για κάθε χρήστη ξεχωριστά, αλλά μια συγκεντρωτική εικόνα της συνολικής απόδοσης του collaborative recommender στο επιλεγμένο δείγμα χρηστών.

```

235 # MAIN
236 if __name__ == "__main__":
237     random_user = np.random.choice(user_ids)
238
239     print(f"\nTop-{TOP_K} recommendations for user {random_user}:")
240     top_k_recs = recommend_top_n(als, interaction_matrix, random_user, movies_df, n=TOP_K)
241     print(top_k_recs)
242
243     print("\nEvaluating collaborative recommender...")
244     results = evaluate_model(
245         sample_size=EVAL_SAMPLE_USERS,
246         top_k=TOP_K,
247         eval_threshold=EVAL_THRESHOLD
248     )
249
250     print("\nEvaluation Results:")
251     for metric, value in results.items():
252         if isinstance(value, float):
253             print(f"{metric}: {value:.4f}")
254         else:
255             print(f"{metric}: {value}")

```

Εικόνα 59: Εκτέλεση και αξιολόγηση collaborative filtering μοντέλου

Στο τελικό τμήμα του collaborative κώδικα ορίζεται η κύρια ροή εκτέλεσης του προγράμματος. Η συνθήκη `if __name__ == "__main__":` εξασφαλίζει ότι το συγκεκριμένο μέρος θα εκτελεστεί μόνο όταν το αρχείο τρέχει ως κύριο script και όχι όταν εισάγεται ως module σε άλλο πρόγραμμα. Αρχικά, επιλέγεται ένας τυχαίος χρήστης από το σύνολο `user_ids` με χρήση της `np.random.choice`, ώστε να παρουσιαστεί ένα ενδεικτικό παράδειγμα λειτουργίας του collaborative recommender πάνω σε πραγματικά δεδομένα.

Επιπροσθέτως, εμφανίζεται ένα ενημερωτικό μήνυμα και καλείται η συνάρτηση `recommend_top_n`, η οποία παράγει τις top-K προτάσεις για τον συγκεκριμένο χρήστη με βάση το εκπαιδευμένο μοντέλο ALS και τη μήτρα αλληλεπιδράσεων. Το αποτέλεσμα αποθηκεύεται στη μεταβλητή `top_k_recs` και στη συνέχεια εκτυπώνεται, ώστε να παρουσιαστεί μια πραγματική λίστα προτάσεων μαζί με τα αντίστοιχα scores. Με αυτόν τον τρόπο, πριν από τη συνολική αξιολόγηση του μοντέλου, δίνεται πρώτα ένα ποιοτικό παράδειγμα του είδους των ταινιών που προτείνει το σύστημα σε έναν μεμονωμένο χρήστη.

Έπειτα, ο κώδικας περνά στο στάδιο της συνολικής αξιολόγησης του collaborative recommender. Εμφανίζεται σχετικό μήνυμα και καλείται η συνάρτηση `evaluate_model`, στην οποία δίνονται ως παράμετροι το μέγεθος του δείγματος χρηστών (`EVAL_SAMPLE_USERS`), το πλήθος προτάσεων ανά χρήστη (`TOP_K`) και το κατώφλι σχετικότητας (`EVAL_THRESHOLD`). Η συνάρτηση επιστρέφει ένα λεξικό `results`, το οποίο περιέχει τις τελικές συγκεντρωτικές μετρικές αξιολόγησης του μοντέλου πάνω στο επιλεγμένο δείγμα χρηστών.

Τέλος, ο κώδικας εκτυπώνει τα αποτελέσματα της αξιολόγησης. Για κάθε στοιχείο του λεξικού `results`, ελέγχεται αν η τιμή είναι δεκαδικός αριθμός· αν ναι, εμφανίζεται μορφοποιημένη με τέσσερα δεκαδικά ψηφία, ενώ διαφορετικά εκτυπώνεται αυτούσια. Με αυτόν τον τρόπο, το τελικό μέρος του προγράμματος συνδυάζει δύο βασικές λειτουργίες: αφενός παρουσιάζει ένα ενδεικτικό παράδειγμα collaborative προτάσεων για έναν τυχαίο χρήστη και αφετέρου παρέχει μια συνολική εικόνα της απόδοσης του μοντέλου μέσω των τελικών μετρικών αξιολόγησης.

4.2.2 Παρουσίαση των αποτελεσμάτων αρχείου collaborative.py

```
Movies raw: (87585, 3)
Train raw: (25474164, 6)
Test raw: (6368541, 6)
Unique movies in original movies.csv: 87585
Unique movies in TRAIN: 31961
Unique movies in TEST: 31445
Candidate movies for recommendation: 31961
Train after alignment: (25474164, 6)
Test after alignment: (6368541, 6)
Final candidate movie count: 31961
Interaction matrix shape: (200948, 31961)
ALS model loaded from disk
```

Εικόνα 60: Αποτελέσματα προεπεξεργασίας ALS μοντέλου

Το αρχικό στάδιο προετοιμασίας των δεδομένων ακολουθεί την ίδια λογική με εκείνη του content-based μοντέλου. Συγκεκριμένα, φορτώνονται τα αρχεία movies, train και test, ορίζεται εκ νέου το candidate catalog μόνο στις ταινίες που εμφανίζονται στο training set και πραγματοποιείται η αντίστοιχη ευθυγράμμιση των δεδομένων. Για τη συγκεκριμένη εκτέλεση, ο αρχικός κατάλογος περιλαμβάνει 87.585 ταινίες, από τις οποίες 31.961 εμφανίζονται στο training set και 31.445 στο test set, ενώ το τελικό candidate catalog περιλαμβάνει 31.961 ταινίες. Παράλληλα, τα σύνολα train και test περιέχουν 25.474.164 και 6.368.541 αξιολογήσεις αντίστοιχα και παραμένουν αμετάβλητα μετά την ευθυγράμμιση με το τελικό recommendation space. Επομένως, και στην collaborative υλοποίηση η αξιολόγηση πραγματοποιείται στο ίδιο train-based candidate catalog, ώστε η σύγκριση με το content-based μοντέλο να παραμένει μεθοδολογικά συνεπής.

Η ουσιαστική διαφοροποίηση της collaborative προσέγγισης εμφανίζεται από το σημείο δημιουργίας της μήτρας αλληλεπιδράσεων χρηστών-ταινιών. Η ένδειξη Interaction matrix shape: (200948, 31961) δείχνει ότι το μοντέλο εκπαιδεύεται πάνω σε πίνακα 200.948 χρηστών και 31.961 ταινιών, όπου κάθε καταγεγραμμένη αλληλεπίδραση εκφράζεται μέσω

των διαθέσιμων ratings. Σε αντίθεση με το content-based μοντέλο, στο οποίο η βασική αναπαράσταση ήταν η μήτρα TF-IDF των χαρακτηριστικών περιεχομένου, εδώ η μάθηση πραγματοποιείται πάνω στη δομή του πίνακα χρηστών-ταινιών, δηλαδή πάνω στα μοτίβα αξιολόγησης που προκύπτουν από τη συλλογική συμπεριφορά των χρηστών. Αυτό αποτελεί και τον πυρήνα της collaborative λογικής, καθώς το μοντέλο δεν βασίζεται σε ρητά περιγραφικά χαρακτηριστικά των ταινιών, αλλά σε λανθάνουσες σχέσεις που μαθαίνει από τις αλληλεπιδράσεις.

Επιπλέον, το μήνυμα ALS model loaded from disk δείχνει ότι για τη συγκεκριμένη εκτέλεση δεν πραγματοποιήθηκε νέα εκπαίδευση του αλγορίθμου, αλλά χρησιμοποιήθηκε ήδη αποθηκευμένο μοντέλο από προηγούμενο στάδιο. Αυτό σημαίνει ότι το ALS είχε εκπαιδευτεί νωρίτερα πάνω στη μήτρα αλληλεπιδράσεων και στη συνέχεια αποθηκεύτηκε σε αρχείο, ώστε να μπορεί να επαναφορτώνεται σε επόμενες εκτελέσεις χωρίς επιπλέον υπολογιστικό κόστος. Η πρακτική αυτή είναι ιδιαίτερα χρήσιμη σε μεγάλα σύνολα δεδομένων, καθώς μειώνει σημαντικά τον χρόνο εκτέλεσης και επιτρέπει την άμεση παραγωγή προτάσεων και αποτελεσμάτων αξιολόγησης χωρίς να απαιτείται κάθε φορά νέα εκπαίδευση του μοντέλου.

Top-10 recommendations for user 67742:

	movieId	title	score
0	589	Terminator 2: Judgment Day (1991)	0.929637
1	1240	Terminator, The (1984)	0.817652
2	110	Braveheart (1995)	0.746959
3	2000	Lethal Weapon (1987)	0.725421
4	1201	Good, the Bad and the Ugly, The (Buono, il bru...	0.714994
5	377	Speed (1994)	0.666790
6	1954	Rocky (1976)	0.650789
7	1291	Indiana Jones and the Last Crusade (1989)	0.644381
8	2916	Total Recall (1990)	0.624828
9	780	Independence Day (a.k.a. ID4) (1996)	0.573657

Εικόνα 61: Top-10 συστάσεις ALS για χρήστη

Το παραπάνω αποτέλεσμα παρουσιάζει τις δέκα κορυφαίες προτάσεις του collaborative μοντέλου ALS για τον χρήστη 67742. Κάθε ταινία συνοδεύεται από ένα score, το οποίο εκφράζει τον βαθμό συνάφειας που αποδίδει το μοντέλο στη συγκεκριμένη πρόταση για τον χρήστη αυτό. Σε αντίθεση με το content-based σύστημα, όπου το score προέκυπτε από

ομοιότητα περιεχομένου, εδώ η τιμή αυτή βασίζεται στη λανθάνουσα αναπαράσταση που έχει μάθει το ALS από τα μοτίβα αξιολόγησης του συνόλου των χρηστών. Επομένως, όσο υψηλότερο είναι το score, τόσο πιο πιθανό θεωρεί το collaborative μοντέλο ότι η συγκεκριμένη ταινία ταιριάζει στις προτιμήσεις του χρήστη, με βάση τη συλλογική συμπεριφορά παρόμοιων χρηστών και παρόμοιων αντικειμένων.

Οι προτάσεις που εμφανίζονται δείχνουν ότι το μοντέλο συγκεντρώνεται κυρίως σε γνωστές και έντονα δημοφιλείς ταινίες δράσης, περιπέτειας και επιστημονικής φαντασίας, όπως οι *Terminator 2: Judgment Day*, *The Terminator*, *Braveheart*, *Lethal Weapon*, *Speed*, *Indiana Jones and the Last Crusade*, *Total Recall* και *Independence Day*. Η εικόνα αυτή είναι χαρακτηριστική της collaborative προσέγγισης, καθώς το μοντέλο δεν βασίζεται άμεσα σε ρητά περιγραφικά χαρακτηριστικά όπως genres, director ή cast, αλλά σε λανθάνουσες σχέσεις που έχουν μάθει από τα ratings. Με άλλα λόγια, το ALS έχει εντοπίσει ότι ο συγκεκριμένος χρήστης βρίσκεται κοντά, στον λανθάνοντα χώρο του μοντέλου, σε άλλους χρήστες που έχουν δείξει προτίμηση σε παρόμοιους τίτλους. Οι αρκετά υψηλές τιμές score, ιδιαίτερα στις πρώτες θέσεις της λίστας, δείχνουν ότι το μοντέλο εμφανίζει σχετικά ισχυρή βεβαιότητα ως προς τη συνάφεια αυτών των προτάσεων. Συνολικά, το συγκεκριμένο output λειτουργεί ως ποιοτική ένδειξη ότι το collaborative recommender είναι σε θέση να παράγει συνεκτικές και αναγνωρίσιμες προτάσεις, βασισμένες όχι στο περιεχόμενο των ταινιών αλλά στη δομή των συλλογικών προτύπων αξιολόγησης που έχει μάθει από τα δεδομένα.

```
Evaluating collaborative recommender...

Evaluation Results:
Users evaluated: 293
Precision@10: 0.2949
Recall@10: 0.2373
MAP@10: 0.2463
NDCG@10: 0.6438
HitRate@10: 0.8703
Diversity: 0.5875
```

Εικόνα 62: Αποτελέσματα αξιολόγησης collaborative filtering

Σε επίπεδο συνολικής αξιολόγησης, το collaborative μοντέλο εξετάστηκε σε 293 χρήστες, δηλαδή σε όσους παρέμειναν τελικά κατάλληλοι για αξιολόγηση μετά τους σχετικούς ελέγχους του pipeline. Για το συγκεκριμένο πείραμα προέκυψαν οι τιμές $\text{Precision@10} = 0.2949$, $\text{Recall@10} = 0.2373$, $\text{MAP@10} = 0.2463$, $\text{NDCG@10} = 0.6438$, $\text{HitRate@10} = 0.8703$ και $\text{Diversity} = 0.5875$. Οι τιμές αυτές αποτυπώνουν τη συνολική απόδοση του collaborative recommender στο επιλεγμένο δείγμα χρηστών και παρέχουν μια συγκεντρωτική εικόνα της συμπεριφοράς του μοντέλου ως προς τη συνάφεια, την ποιότητα κατάταξης και την ποικιλία των προτάσεων. Η αναλυτική ερμηνεία των μετρικών αυτών και η διεξοδική αξιολόγησή τους παρουσιάζονται σε επόμενο κεφάλαιο.

4.2.3 Αξιολόγηση Μοντέλου

Στο τελευταίο στάδιο της εκτέλεσης πραγματοποιείται η αξιολόγηση του collaborative συστήματος προτάσεων σε τυχαίο δείγμα χρηστών από το test set. Για τη συγκεκριμένη εκτέλεση επιλέχθηκαν αρχικά 300 χρήστες, από τους οποίους τελικά αξιολογήθηκαν 293, καθώς ορισμένοι χρήστες δεν διέθεταν relevant αντικείμενα στο test set ή δεν ήταν δυνατό να παραχθούν έγκυρες προτάσεις για αυτούς. Για κάθε χρήστη το μοντέλο παράγει τις τοπ 10 προτάσεις και τις συγκρίνει με τις relevant ταινίες του test set, δηλαδή με τις ταινίες που έχουν βαθμολογία μεγαλύτερη ή ίση από το κατώφλι $\text{EVAL_THRESHOLD} = 3.5$. Με βάση αυτή τη διαδικασία υπολογίζονται οι μετρικές Precision@10 , Recall@10 , MAP@10 , NDCG@10 , HitRate@10 και Diversity , ενώ στο τέλος εμφανίζεται ο μέσος όρος τους στο σύνολο των αξιολογημένων χρηστών. Για τη συγκεκριμένη εκτέλεση προέκυψαν οι τιμές:

- $\text{Precision@10} = 0.2949$
- $\text{Recall@10} = 0.2373$
- $\text{MAP@10} = 0.2463$
- $\text{NDCG@10} = 0.6438$
- $\text{HitRate@10} = 0.8703$
- $\text{Diversity} = 0.5875$

Η μετρική Precision@10 δείχνει τι ποσοστό από τις δέκα προτάσεις του συστήματος είναι πράγματι relevant για τον χρήστη. Στη συγκεκριμένη περίπτωση, η τιμή 0.2949 σημαίνει ότι κατά μέσο όρο περίπου 2,95 από τις 10 προτάσεις είναι σχετικές με τα relevant αντικείμενα του test set. Ουσιαστικά, η μετρική αυτή δείχνει πόσο «καθαρή» είναι η τελική λίστα

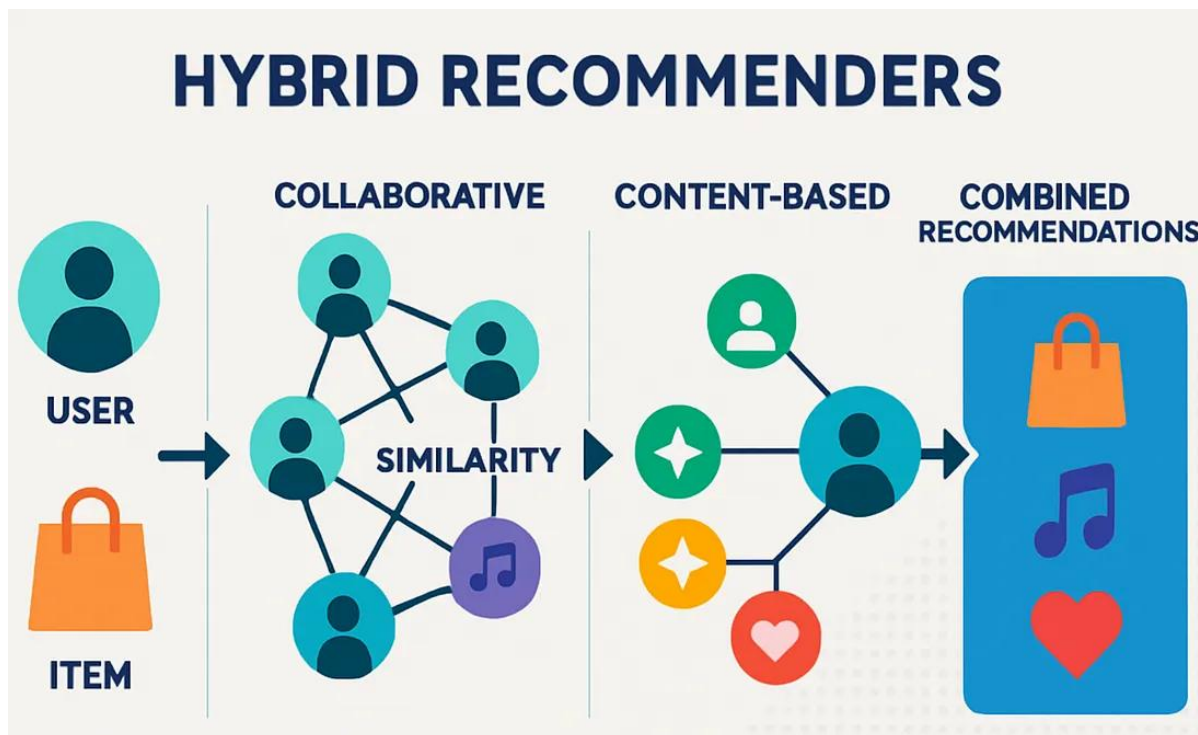
προτάσεων, δηλαδή πόσες από τις πρώτες θέσεις καταλαμβάνονται από σωστές επιλογές. Η Recall@10 , από την άλλη πλευρά, δείχνει πόσο μεγάλο μέρος των συνολικά relevant ταινιών του χρήστη καταφέρνει να ανακτήσει το μοντέλο μέσα στις πρώτες δέκα θέσεις. Εδώ η τιμή 0.2373 σημαίνει ότι το collaborative σύστημα εντοπίζει περίπου το 23,7% των relevant αντικειμένων του χρήστη στο top-10. Με άλλα λόγια, το Precision δείχνει πόσο στοχευμένες είναι οι προτάσεις, ενώ το Recall δείχνει πόσο καλά καλύπτεται το σύνολο των σωστών επιλογών. Το γεγονός ότι και οι δύο μετρικές έχουν σχετικά υψηλές τιμές δείχνει ότι το μοντέλο όχι μόνο προτείνει αρκετά σχετικές ταινίες, αλλά καταφέρνει και να ανακτά ικανοποιητικό μέρος των relevant αντικειμένων του χρήστη.

Η $\text{MAP@10} = 0.2463$ αποτυπώνει τη μέση ακρίβεια της κατάταξης και εξετάζει όχι μόνο αν μια σχετική ταινία προτάθηκε, αλλά και σε ποια θέση εμφανίστηκε. Αυτό σημαίνει ότι relevant ταινίες που τοποθετούνται νωρίς στη λίστα ενισχύουν περισσότερο τη μετρική από relevant ταινίες που εμφανίζονται χαμηλότερα. Επομένως, η συγκεκριμένη τιμή δείχνει ότι το μοντέλο δεν εντοπίζει απλώς σωστές ταινίες, αλλά σε αρκετές περιπτώσεις τις τοποθετεί και σχετικά ψηλά στην τελική κατάταξη. Με παρόμοια λογική, η $\text{NDCG@10} = 0.6438$ αξιολογεί επίσης την ποιότητα της κατάταξης, δίνοντας ιδιαίτερη βαρύτητα στις πρώτες θέσεις της λίστας. Η τιμή αυτή είναι αρκετά υψηλή και δείχνει ότι οι relevant προτάσεις τείνουν να εμφανίζονται κοντά στην κορυφή του recommendation list, γεγονός ιδιαίτερα σημαντικό στην πράξη, αφού ο χρήστης συνήθως δίνει μεγαλύτερη προσοχή στις πρώτες προτάσεις που βλέπει. Η $\text{HitRate@10} = 0.8703$ είναι από τις πιο εύκολα ερμηνεύσιμες και πρακτικά χρήσιμες μετρικές. Δείχνει το ποσοστό των χρηστών για τους οποίους το σύστημα κατάφερε να εμφανίσει έστω και μία σχετική ταινία μέσα στις πρώτες δέκα προτάσεις. Η τιμή 0.8703 σημαίνει ότι περίπου στο 87% των αξιολογημένων χρηστών το collaborative recommender πέτυχε τουλάχιστον ένα «χτύπημα», δηλαδή τουλάχιστον μία χρήσιμη πρόταση. Αυτό αποτελεί πολύ σημαντική ένδειξη πρακτικής αποτελεσματικότητας, καθώς δείχνει ότι το σύστημα λειτουργεί ικανοποιητικά για τη μεγάλη πλειονότητα των χρηστών και δεν αποτυγχάνει συστηματικά να εντοπίσει σχετικές ταινίες. Τέλος, η μετρική $\text{Diversity} = 0.5875$ δείχνει τον βαθμό ποικιλίας μέσα στην ίδια τη λίστα προτάσεων. Σε αυτή την περίπτωση η diversity υπολογίζεται πάνω στους latent παράγοντες των αντικειμένων που έχει μάθει το ALS και εκφράζει το πόσο διαφορετικές είναι μεταξύ τους οι προτεινόμενες ταινίες. Η τιμή αυτή δείχνει ότι το μοντέλο παράγει λίστες με κάποιον βαθμό ποικιλίας, αλλά όχι σε τόσο υψηλό επίπεδο όσο ένα πιο περιγραφικά διαφοροποιημένο content-based σύστημα. Αυτό είναι

αναμενόμενο, καθώς το collaborative filtering συχνά τείνει να συγκεντρώνει τις προτάσεις σε αντικείμενα που είναι πολύ κοντά μεταξύ τους στον λανθάνοντα χώρο προτιμήσεων.

Συνολικά, τα αποτελέσματα δείχνουν ότι το collaborative μοντέλο παρουσιάζει ισχυρή συμπεριφορά ως προς τη συνάφεια και την ποιότητα της κατάταξης των προτάσεων. Οι σχετικές τιμές των Precision@10, Recall@10, MAP@10, NDCG@10 και HitRate@10 δείχνουν ότι το ALS καταφέρνει να εντοπίζει relevant ταινίες με αρκετά αποτελεσματικό τρόπο και να τις εμφανίζει σε υψηλές θέσεις της λίστας. Παράλληλα, η τιμή Diversity = 0.5875 δείχνει ότι το collaborative μοντέλο διατηρεί έναν ικανοποιητικό βαθμό ποικιλίας στις τελικές προτάσεις. Οι ταινίες που εμφανίζονται στη λίστα δεν είναι υπερβολικά όμοιες μεταξύ τους, αλλά παραμένουν μέσα σε έναν σχετικά συνεκτικό χώρο προτίμησης, κάτι που είναι αναμενόμενο για collaborative προσέγγιση. Επομένως, η συνολική εικόνα του evaluation δείχνει ότι το collaborative recommender παρουσιάζει ισχυρή συμπεριφορά ως προς τη συνάφεια και την αποτελεσματικότητα των προτάσεων, διατηρώντας ταυτόχρονα έναν ικανοποιητικό βαθμό εσωτερικής διαφοροποίησης στις τελικές λίστες.

4.3 Hybrid Filtering



Εικόνα 63: Hybrid Filtering, Πηγή: medium.com

Τα υβριδικά συστήματα σύστασης, ή hybrid recommender systems, αποτελούν προσεγγίσεις που συνδυάζουν δύο ή περισσότερες τεχνικές σύστασης με σκοπό την αξιοποίηση των πλεονεκτημάτων κάθε μεθόδου και τον περιορισμό των αδυναμιών τους. Στη βιβλιογραφία, οι βασικές κατηγορίες συστημάτων σύστασης διακρίνονται συνήθως σε content-based, collaborative και hybrid προσεγγίσεις. Τα content-based συστήματα βασίζονται στα χαρακτηριστικά των αντικειμένων και στο ιστορικό προτιμήσεων του χρήστη, ενώ τα collaborative συστήματα αξιοποιούν τα μοτίβα αλληλεπίδρασης πολλών χρηστών με τα αντικείμενα. Τα hybrid συστήματα προσπαθούν να συνδυάσουν αυτές τις δύο λογικές, ώστε η τελική πρόταση να μη βασίζεται αποκλειστικά ούτε στο περιεχόμενο ούτε στη συλλογική συμπεριφορά των χρηστών (Adomavicius & Tuzhilin, 2005).

Η ανάγκη για υβριδικές προσεγγίσεις προκύπτει από τους περιορισμούς των επιμέρους μεθόδων. Τα content-based συστήματα μπορούν να προτείνουν αντικείμενα παρόμοια με αυτά που έχει ήδη προτιμήσει ο χρήστης, όμως συχνά δυσκολεύονται να προτείνουν αντικείμενα που αποκλίνουν από το υπάρχον προφίλ του, οδηγώντας σε περιορισμένη ποικιλία ή υπερβολική εξειδίκευση. Από την άλλη πλευρά, τα collaborative συστήματα μπορούν να ανακαλύψουν σχέσεις που δεν είναι εμφανείς από τα περιγραφικά χαρακτηριστικά των

αντικειμένων, αλλά επηρεάζονται από προβλήματα όπως η αραιότητα των δεδομένων και το cold-start, ιδιαίτερα όταν υπάρχουν νέοι χρήστες ή νέα αντικείμενα με λίγες αξιολογήσεις. Τα hybrid recommender systems επιχειρούν να αντιμετωπίσουν αυτούς τους περιορισμούς, συνδυάζοντας διαφορετικές πηγές πληροφορίας σε ένα ενιαίο σύστημα (Burke, 2002).

Στην παρούσα εργασία υλοποιείται ένα weighted hybrid recommender, δηλαδή ένα υβριδικό μοντέλο στο οποίο το τελικό score κάθε ταινίας προκύπτει ως γραμμικός συνδυασμός ενός collaborative score και ενός content-based score. Το collaborative μέρος βασίζεται σε ALS matrix factorization και αξιοποιεί τη μήτρα αλληλεπιδράσεων χρηστών-ταινιών, ενώ το content-based μέρος βασίζεται σε TF-IDF αναπαράσταση εμπλουτισμένων μεταδεδομένων των ταινιών και cosine similarity με το προφίλ του χρήστη. Πριν από τον τελικό συνδυασμό, τα δύο scores κανονικοποιούνται ώστε να βρίσκονται σε κοινή κλίμακα. Το τελικό hybrid score υπολογίζεται με βάση τον τύπο:

$$final_score = \alpha \cdot collab_score + (1 - \alpha) \cdot content_score$$

όπου το α καθορίζει το βάρος του collaborative μέρους. Στη συγκεκριμένη υλοποίηση έχει οριστεί $\alpha = 0.70$, επομένως το τελικό score βασίζεται κατά 70% στο collaborative component και κατά 30% στο content-based component. Η μορφή αυτή αντιστοιχεί στη λογική του weighted hybridization, όπου τα αποτελέσματα διαφορετικών recommenders συνδυάζονται μέσω σταθμισμένου αθροίσματος. Στη βιβλιογραφία περιγράφονται τέτοιου τύπου υβριδικές προσεγγίσεις ως περιπτώσεις όπου διαφορετικοί recommenders παράγουν επιμέρους αποτελέσματα και το σύστημα τα συνδυάζει για να παραγάγει την τελική σύσταση (Cremonesi, Turrin, and Airolidi 2011).

4.3.1 Ανάλυση αρχείου hybrid.py

```
1  import os
2  import re
3  import pickle
4  from pathlib import Path
5  import numpy as np
6  import pandas as pd
7  from scipy.sparse import coo_matrix
8  from implicit.als import AlternatingLeastSquares
9  from sklearn.feature_extraction.text import TfidfVectorizer
10 from sklearn.metrics import ndcg_score
11 from sklearn.metrics.pairwise import cosine_similarity
12
13 # SETTINGS
14 TOP_K = 10
15 EVAL_SAMPLE_USERS = 300
16 PROFILE_THRESHOLD = 4.0
17 EVAL_THRESHOLD = 3.5
18 WEIGHT_MODE = "discrete_strong"
19 ALPHA = 0.70
20
21 BASE_DIR = Path(__file__).resolve().parent.parent
22
23 movies_path = BASE_DIR / "data" / "ml-32m" / "movies.csv"
24 tags_path = BASE_DIR / "data" / "ml-32m" / "tags.csv"
25 train_path = BASE_DIR / "data" / "ml-32m" / "train.csv"
26 test_path = BASE_DIR / "data" / "ml-32m" / "test.csv"
27 tmdb_path = BASE_DIR / "data" / "ml-32m" / "tmdb_enriched_movies.csv"
28 model_path = BASE_DIR / "data" / "ml-32m" / "als_model_new.pkl"
29
30 os.environ["OPENBLAS_NUM_THREADS"] = "1"
```

Εικόνα 64: Ρυθμίσεις hybrid συστήματος συστάσεων

Στο αρχικό τμήμα του hybrid recommender πραγματοποιείται η εισαγωγή των βιβλιοθηκών, ο ορισμός των βασικών παραμέτρων και η δήλωση των διαδρομών των αρχείων που θα χρησιμοποιηθούν. Επειδή το συγκεκριμένο μοντέλο συνδυάζει content-based και collaborative filtering, οι βιβλιοθήκες που εισάγονται καλύπτουν και τις δύο πλευρές της υβριδικής προσέγγισης. Συγκεκριμένα, οι numpy και pandas χρησιμοποιούνται για

αριθμητικές πράξεις και διαχείριση των δεδομένων, η `coo_matrix` για τη δημιουργία της sparse μήτρας αλληλεπιδράσεων χρηστών-ταινιών, ενώ η `AlternatingLeastSquares` χρησιμοποιείται για το collaborative κομμάτι του μοντέλου μέσω ALS. Παράλληλα, ο `TfidfVectorizer` χρησιμοποιείται για τη δημιουργία της TF-IDF αναπαράστασης των ταινιών στο content-based μέρος, ενώ η `cosine_similarity` αξιοποιείται τόσο για τον υπολογισμό ομοιότητας περιεχομένου όσο και για τη μέτρηση της diversity. Η `ndcg_score` χρησιμοποιείται στο στάδιο της αξιολόγησης, ενώ οι βιβλιοθήκες `os`, `pickle`, `re` και `Path` εξυπηρετούν βοηθητικές λειτουργίες, όπως ο καθαρισμός κειμένου, η διαχείριση διαδρομών, η αποθήκευση ή φόρτωση του ALS μοντέλου και η ρύθμιση του περιβάλλοντος εκτέλεσης.

Έπειτα, ορίζονται οι βασικές παράμετροι του συστήματος. Η μεταβλητή `TOP_K = 10` καθορίζει ότι το σύστημα θα επιστρέφει δέκα προτάσεις ανά χρήστη, ενώ η `EVAL_SAMPLE_USERS = 300` ορίζει το πλήθος των χρηστών που θα χρησιμοποιηθεί στην αξιολόγηση. Το `PROFILE_THRESHOLD = 4.0` καθορίζει ποιες ταινίες θεωρούνται θετικές για την κατασκευή του content-based προφίλ του χρήστη, ενώ το `EVAL_THRESHOLD = 3.5` καθορίζει ποιες ταινίες θεωρούνται relevant κατά την αξιολόγηση στο test set. Η μεταβλητή `WEIGHT_MODE = "discrete_strong"` δηλώνει ότι οι θετικές αξιολογήσεις δεν συμμετέχουν όλες με το ίδιο βάρος στο content profile, αλλά οι υψηλότερες βαθμολογίες αποκτούν μεγαλύτερη επιρροή. Η σημαντικότερη παράμετρος για την υβριδική λογική είναι η `ALPHA = 0.70`, η οποία καθορίζει το βάρος του collaborative component στο τελικό score. Συγκεκριμένα, το τελικό hybrid score θα αποτελείται κατά 70% από το collaborative score και κατά 30% από το content-based score, δίνοντας έτσι μεγαλύτερη έμφαση στα μοτίβα συλλογικής συμπεριφοράς, χωρίς όμως να αγνοείται η περιγραφική πληροφορία των ταινιών.

Τέλος, ορίζεται το `BASE_DIR`, το οποίο επιτρέπει τη δημιουργία σχετικών διαδρομών ανεξάρτητα από το σημείο εκτέλεσης του αρχείου. Με βάση αυτό, δηλώνονται οι διαδρομές προς τα βασικά αρχεία του MovieLens και του εμπλουτισμού TMDb: `movies.csv`, `tags.csv`, `train.csv`, `test.csv` και `tmdb_enriched_movies.csv`. Επιπλέον, η μεταβλητή `model_path` δείχνει στο αποθηκευμένο ALS μοντέλο, το οποίο χρησιμοποιείται από το collaborative τμήμα του hybrid recommender. Η τελευταία εντολή, `os.environ["OPENBLAS_NUM_THREADS"] = "1"`, χρησιμοποιείται για να περιορίσει το OpenBLAS ώστε να εκτελεί τις αριθμητικές πράξεις του με ένα μόνο νήμα επεξεργασίας. Το OpenBLAS είναι μια βιβλιοθήκη που χρησιμοποιείται εσωτερικά από εργαλεία αριθμητικών υπολογισμών, όπως η `numpy` και άλλες βιβλιοθήκες που

βασίζονται σε πράξεις πινάκων. Κατά την εκπαίδευση ή τη χρήση του ALS εκτελούνται πολλές πράξεις γραμμικής άλγεβρας πάνω σε μεγάλους πίνακες και sparse matrices. Σε ορισμένα περιβάλλοντα εκτέλεσης, η ταυτόχρονη χρήση πολλών νημάτων από το OpenBLAS μπορεί να δημιουργήσει προβλήματα σταθερότητας, υπερβολική κατανάλωση πόρων ή καθυστερήσεις λόγω ανταγωνισμού μεταξύ των threads. Για τον λόγο αυτό, με τη συγκεκριμένη εντολή ορίζεται ότι οι πράξεις αυτές θα εκτελούνται με ένα νήμα, ώστε η εκτέλεση του προγράμματος να είναι πιο σταθερή και προβλέψιμη. Η εντολή αυτή δεν αλλάζει τη λογική του recommender ούτε επηρεάζει τα αποτελέσματα του μοντέλου· αφορά αποκλειστικά τον τρόπο με τον οποίο το υπολογιστικό περιβάλλον διαχειρίζεται τους αριθμητικούς υπολογισμούς. Συνολικά, το αρχικό αυτό τμήμα θέτει το πλαίσιο της υβριδικής υλοποίησης, συνδυάζοντας τις απαραίτητες δομές για content-based αναπαράσταση, collaborative learning και τελική αξιολόγηση.

```

31 #HELPERS
32 def clean_text(text): 4 usages
33     text = str(text).lower().strip()
34     text = re.sub(pattern=r"[^a-z0-9\s]", repl: " ", text)
35     text = re.sub(pattern=r"\s+", repl: " ", text).strip()
36     return text
37 def truncate_words(text, max_words=60): 1 usage
38     words = str(text).split()
39     return " ".join(words[:max_words])
40 def get_rating_weights(ratings_array, mode="discrete"): 1 usage
41     if mode == "linear":
42         weights = ratings_array - PROFILE_THRESHOLD + 1.0
43     elif mode == "raw":
44         weights = ratings_array
45     elif mode == "squared":
46         weights = (ratings_array - 3.5) ** 2
47     elif mode == "discrete":
48         weights = np.where(
49             ratings_array >= 5.0, 3.0,
50             np.where(ratings_array >= 4.5, 2.0, 1.0)
51         )
52     elif mode == "discrete_strong":
53         weights = np.where(
54             ratings_array >= 5.0, 4.0,
55             np.where(ratings_array >= 4.5, 2.5, 1.0)
56         )
57     else:
58         raise ValueError(f"Unknown WEIGHT_MODE: {mode}")
59
60     return weights.reshape(-1, 1)

```

Εικόνα 65: Βοηθητικές συναρτήσεις του hybrid συστήματος

Στο επόμενο τμήμα ορίζονται οι πρώτες βοηθητικές συναρτήσεις του hybrid recommender, οι οποίες χρησιμοποιούνται κυρίως για τον καθαρισμό των κειμενικών δεδομένων και για τη στάθμιση των αξιολογήσεων που συμμετέχουν στη δημιουργία του content-based προφίλ χρήστη. Η συνάρτηση `clean_text` δέχεται ως είσοδο ένα κείμενο και το

μετατρέπει αρχικά σε συμβολοσειρά, ώστε να μπορεί να επεξεργαστεί ακόμη και τιμές που δεν είναι ήδη σε μορφή string. Στη συνέχεια το μετατρέπει σε πεζά γράμματα και αφαιρεί τα περιττά κενά από την αρχή και το τέλος. Με την πρώτη `re.sub`, αντικαθιστά όλους τους χαρακτήρες που δεν είναι αγγλικά γράμματα, αριθμοί ή κενά με κενό διάστημα. Έτσι αφαιρούνται σημεία στίξης και ειδικοί χαρακτήρες που θα μπορούσαν να δημιουργήσουν θόρυβο στη μεταγενέστερη TF-IDF αναπαράσταση. Με τη δεύτερη `re.sub`, πολλαπλά συνεχόμενα κενά αντικαθίστανται από ένα μόνο κενό και εφαρμόζεται ξανά `strip`, ώστε το τελικό αποτέλεσμα να είναι καθαρό και ομοιόμορφο. Η συνάρτηση αυτή είναι σημαντική, επειδή στο hybrid μοντέλο το content-based μέρος στηρίζεται σε λεκτικά χαρακτηριστικά, όπως τίτλοι, genres, tags, keywords, director και cast, τα οποία πρέπει να βρίσκονται σε ενιαία μορφή πριν μετατραπούν σε TF-IDF διανύσματα.

Η συνάρτηση `truncate_words` χρησιμοποιείται για τον περιορισμό του μήκους μεγάλων κειμενικών πεδίων. Αρχικά μετατρέπει το κείμενο σε string και το χωρίζει σε επιμέρους λέξεις με τη `split()`. Στη συνέχεια κρατά μόνο τις πρώτες `max_words` λέξεις, με προεπιλεγμένη τιμή τις 60, και τις ενώνει ξανά σε ένα ενιαίο string. Η λειτουργία αυτή είναι χρήσιμη κυρίως για πεδία όπως το `overview`, τα οποία μπορεί να είναι πολύ μεγάλα και να επηρεάσουν δυσανάλογα τη διανυσματική αναπαράσταση της ταινίας. Με τον περιορισμό των λέξεων διατηρείται η βασική περιγραφική πληροφορία, χωρίς ένα μεγάλο κείμενο να κυριαρχεί υπερβολικά στο τελικό `rich_content`.

Η συνάρτηση `get_rating_weights` καθορίζει με ποιο βάρος θα συμμετέχουν οι θετικά αξιολογημένες ταινίες στη δημιουργία του content-based προφίλ του χρήστη. Η είσοδος της είναι ένας πίνακας βαθμολογιών και ένας τρόπος στάθμισης (`mode`). Στον ορισμό της συνάρτησης η προεπιλεγμένη τιμή είναι `mode="discrete"`, κάτι που σημαίνει ότι αν η συνάρτηση κληθεί χωρίς να δοθεί διαφορετική τιμή για το `mode`, τότε θα εφαρμοστεί η απλή `discrete` στάθμιση. Ωστόσο, στη συγκεκριμένη υβριδική υλοποίηση η συνάρτηση καλείται με `mode=WEIGHT_MODE`, και επειδή στην αρχή του κώδικα έχει οριστεί `WEIGHT_MODE = "discrete_strong"`, η στάθμιση που εφαρμόζεται τελικά είναι η `discrete_strong`. Επομένως, το `discrete` λειτουργεί ως default επιλογή ασφαλείας, ενώ η πραγματική επιλογή του συγκεκριμένου πειράματος είναι η ισχυρότερη διακριτή στάθμιση. Αυτό σημαίνει ότι οι θετικές αξιολογήσεις δεν αντιμετωπίζονται όλες ισότιμα. Συγκεκριμένα, οι ταινίες με βαθμολογία 5.0 παίρνουν βάρος 4.0, οι ταινίες με βαθμολογία τουλάχιστον 4.5 παίρνουν βάρος

2.5, ενώ οι υπόλοιπες θετικές ταινίες, δηλαδή όσες έχουν περάσει το PROFILE_THRESHOLD, παίρνουν βάρος 1.0. Με αυτόν τον τρόπο, οι ταινίες που άρεσαν περισσότερο στον χρήστη επηρεάζουν πιο έντονα το τελικό content profile, ενώ οι απλώς θετικές αξιολογήσεις συμμετέχουν με μικρότερη βαρύτητα.

```

62 def minmax_1d(x): 2 usages
63     x = np.asarray(x, dtype=np.float32)
64     if x.size == 0:
65         return x
66     x_min = np.min(x)
67     x_max = np.max(x)
68     if x_max - x_min < 1e-12:
69         return np.zeros_like(x, dtype=np.float32)
70     return (x - x_min) / (x_max - x_min)
71
72 def precision_at_k(recommended_items, relevant_items, k): 1 usage
73     hits = sum(1 for item in recommended_items[:k] if item in relevant_items)
74     return hits / k if k > 0 else 0.0
75
76 def recall_at_k(recommended_items, relevant_items, k): 1 usage
77     if len(relevant_items) == 0:
78         return 0.0
79     hits = sum(1 for item in recommended_items[:k] if item in relevant_items)
80     return hits / len(relevant_items)
81
82 def average_precision_at_k(recommended_items, relevant_items, k): 1 usage
83     hits = 0
84     precision_sum = 0.0
85     for i, item in enumerate(recommended_items[:k], start=1):
86         if item in relevant_items:
87             hits += 1
88             precision_sum += hits / i
89     denom = min(len(relevant_items), k)
90     return precision_sum / denom if denom > 0 else 0.0

```

Εικόνα 66: Κανονικοποίηση και μετρικές αξιολόγησης hybrid μοντέλου

Η συνάρτηση minmax_1d χρησιμοποιείται για να φέρει ένα σύνολο αριθμητικών τιμών στην ίδια κλίμακα, από 0 έως 1. Αυτό είναι απαραίτητο στο hybrid μοντέλο, επειδή στη συνέχεια θα συνδυαστούν δύο διαφορετικά scores: το collaborative score που προέρχεται από το ALS και το content-based score που προέρχεται από την ομοιότητα περιεχομένου. Τα δύο αυτά scores δεν είναι απαραίτητο να έχουν τις ίδιες τιμές ή το ίδιο εύρος. Για παράδειγμα, το

ένα μπορεί να κυμαίνεται από 0.2 έως 0.9 και το άλλο από 5 έως 30. Αν συνδυάζονταν απευθείας, το δεύτερο θα επηρέαζε πολύ περισσότερο το τελικό αποτέλεσμα απλώς επειδή έχει μεγαλύτερους αριθμούς. Η `minmax_1d` αποτρέπει αυτό το πρόβλημα, μετατρέποντας κάθε σύνολο `scores` σε κοινή κλίμακα. Αρχικά μετατρέπει την είσοδο σε πίνακα `numpy`, ελέγχει αν είναι κενή και στη συνέχεια βρίσκει τη μικρότερη και τη μεγαλύτερη τιμή. Αν όλες οι τιμές είναι ίδιες ή σχεδόν ίδιες, επιστρέφει μηδενικά, γιατί δεν υπάρχει πραγματική διαφορά μεταξύ των `scores` για να κανονικοποιηθεί. Διαφορετικά, εφαρμόζει τον τύπο $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$. Με αυτόν τον τρόπο, η μικρότερη τιμή γίνεται 0, η μεγαλύτερη γίνεται 1 και όλες οι υπόλοιπες τοποθετούνται αναλογικά ανάμεσα στο 0 και το 1. Έτσι, όταν αργότερα υπολογίζεται το τελικό `hybrid score`, το `collaborative` και το `content-based` μέρος συνδυάζονται πιο δίκαια, χωρίς κάποιο από τα δύο να κυριαρχεί τεχνητά λόγω διαφορετικής αριθμητικής κλίμακας.

Παράλληλα ορίζονται οι βασικές μετρικές αξιολόγησης που χρησιμοποιούνται για την αποτίμηση των προτάσεων. Η συνάρτηση `precision_at_k` υπολογίζει πόσες από τις πρώτες `k` προτεινόμενες ταινίες είναι πράγματι `relevant`. Ο κώδικας μετρά τα `hits`, δηλαδή τις προτεινόμενες ταινίες που βρίσκονται μέσα στο σύνολο `relevant_items`, και διαιρεί αυτό το πλήθος με το `k`. Έτσι, η `Precision@K` δείχνει πόσο ακριβής είναι η λίστα προτάσεων στις πρώτες θέσεις της. Η συνάρτηση `recall_at_k` εξετάζει κάτι διαφορετικό: μετρά πόσα από όλα τα `relevant` αντικείμενα του χρήστη κατάφερε να βρει το σύστημα μέσα στις πρώτες `k` προτάσεις. Αν δεν υπάρχουν `relevant` αντικείμενα, επιστρέφεται 0.0, ενώ διαφορετικά τα `hits` διαιρούνται με το συνολικό πλήθος των `relevant` αντικειμένων. Με αυτόν τον τρόπο, το `Recall@K` δείχνει τον βαθμό κάλυψης των πραγματικά σχετικών επιλογών του χρήστη.

Συνεχίζοντας, η `average_precision_at_k` αξιολογεί όχι μόνο αν υπάρχουν `relevant` αντικείμενα στη λίστα, αλλά και σε ποιες θέσεις εμφανίζονται. Η συνάρτηση διατρέπει τις πρώτες `k` προτάσεις και κάθε φορά που βρίσκει `relevant` αντικείμενο αυξάνει τα `hits` και προσθέτει στο `precision_sum` την ακρίβεια μέχρι τη συγκεκριμένη θέση. Αυτό σημαίνει ότι μια `relevant` ταινία που εμφανίζεται πρώτη ή δεύτερη στη λίστα συνεισφέρει περισσότερο από μια `relevant` ταινία που εμφανίζεται χαμηλότερα. Στο τέλος, το άθροισμα κανονικοποιείται με βάση το `min(len(relevant_items), k)`, δηλαδή το μέγιστο πλήθος `relevant` αντικειμένων που θα μπορούσαν να εμφανιστούν μέσα στο `top-k`. Η συνάρτηση αυτή χρησιμοποιείται στη συνέχεια για τον υπολογισμό της `MAP@K`, η οποία προκύπτει ως ο μέσος όρος των `Average Precision` τιμών όλων των χρηστών.

```

92 def ndcg_at_k(recommended_items, scores, relevant_items, k): 1usage
93     y_true = np.array([[1 if item in relevant_items else 0 for item in recommended_items[:k]]])
94     y_score = np.array([scores[:k]])
95     return ndcg_score(y_true, y_score)
96 def hit_rate_at_k(recommended_items, relevant_items, k): 1usage
97     return 1.0 if any(item in relevant_items for item in recommended_items[:k]) else 0.0
98 def intra_list_diversity(recommended_items): 1usage
99     """
100     Diversity measured in content space (TF-IDF), so it stays comparable
101     with the content-based model's diversity definition.
102     """
103     if len(recommended_items) < 2:
104         return 0.0
105     item_indices = [movie_id_to_idx[item] for item in recommended_items if item in movie_id_to_idx]
106
107     if len(item_indices) < 2:
108         return 0.0
109
110     item_vectors = tfidf_matrix[item_indices]
111     sim_matrix = cosine_similarity(item_vectors)
112
113     n = len(item_indices)
114     pairwise_sims = []
115
116     for i in range(n):
117         for j in range(i + 1, n):
118             pairwise_sims.append(sim_matrix[i, j])
119     if not pairwise_sims:
120         return 0.0
121     return 1.0 - np.mean(pairwise_sims)

```

Εικόνα 67: Μετρικές κατάταξης και ποικιλομορφίας hybrid μοντέλου

Σε αυτό το τμήμα συνεχίζεται ο ορισμός των μετρικών αξιολόγησης που χρησιμοποιούνται στο hybrid recommender. Η συνάρτηση `ndcg_at_k` υπολογίζει τη μετρική $NDCG@K$, η οποία αξιολογεί την ποιότητα της κατάταξης των προτάσεων. Αρχικά δημιουργείται το `y_true`, δηλαδή ένας δυαδικός πίνακας που δείχνει για κάθε μία από τις πρώτες `k` προτάσεις αν είναι *relevant* ή όχι. Αν το προτεινόμενο `movieId` υπάρχει μέσα στο σύνολο `relevant_items`, τότε η αντίστοιχη τιμή είναι 1, διαφορετικά είναι 0. Στη συνέχεια δημιουργείται το `y_score`, το οποίο περιέχει τα `scores` των πρώτων `k` προτάσεων. Στο hybrid μοντέλο αυτά τα `scores` αντιστοιχούν στο τελικό `final_score`, δηλαδή στο `score` που προκύπτει από τον συνδυασμό collaborative και content-based συνιστώσας. Τα δύο αυτά στοιχεία δίνονται στη συνάρτηση `ndcg_score`, η οποία υπολογίζει κατά πόσο οι *relevant* ταινίες εμφανίζονται ψηλά στη λίστα. Με απλά λόγια, η $NDCG@K$ δεν εξετάζει μόνο αν υπάρχουν σωστές προτάσεις, αλλά και αν το μοντέλο τις έχει τοποθετήσει σε καλές θέσεις μέσα στην κατάταξη.

Η συνάρτηση `hit_rate_at_k` υπολογίζει το `HitRate@K` με τον δυαδικό ορισμό που αναφέρθηκε και στο θεωρητικό μέρος της αξιολόγησης. Συγκεκριμένα, ελέγχει αν υπάρχει έστω μία *relevant* ταινία μέσα στις πρώτες *k* προτάσεις. Αν υπάρχει τουλάχιστον ένα τέτοιο αντικείμενο, επιστρέφεται η τιμή 1.0, ενώ αν δεν υπάρχει κανένα επιστρέφεται 0.0. Άρα, για κάθε χρήστη, η μετρική αυτή δεν μετρά πόσες σωστές προτάσεις εμφανίστηκαν, αλλά μόνο αν υπήρξε τουλάχιστον μία επιτυχία μέσα στο *top-k*. Στη συνέχεια, κατά την αξιολόγηση του μοντέλου, οι τιμές αυτές υπολογίζονται για όλους τους χρήστες και ο μέσος όρος τους δείχνει το ποσοστό των χρηστών για τους οποίους το σύστημα κατάφερε να εμφανίσει τουλάχιστον μία σχετική πρόταση.

Εν κατακλείδι, η συνάρτηση `intra_list_diversity` υπολογίζει την εσωτερική ποικιλία της λίστας προτάσεων. Αρχικά, η συνάρτηση ελέγχει αν η λίστα περιέχει τουλάχιστον δύο προτεινόμενες ταινίες, καθώς η ποικιλία δεν μπορεί να υπολογιστεί με ένα μόνο αντικείμενο. Στη συνέχεια, τα `movieId` των προτεινόμενων ταινιών μετατρέπονται στα αντίστοιχα `indices` της `tfidf_matrix` μέσω του `movie_id_to_idx`. Αν μετά από αυτόν τον έλεγχο υπάρχουν λιγότερες από δύο έγκυρες ταινίες, επιστρέφεται ξανά 0.0. Έπειτα, ανακτώνται από τη `tfidf_matrix` τα διανύσματα των προτεινόμενων ταινιών και υπολογίζεται ο πίνακας `cosine similarity` μεταξύ τους. Ο κώδικας στη συνέχεια εξετάζει όλα τα μοναδικά ζεύγη ταινιών της λίστας με τους δύο βρόχους `for`. Η εσωτερική επανάληψη ξεκινά από το `i + 1`, ώστε να αποφεύγεται τόσο η σύγκριση μιας ταινίας με τον εαυτό της όσο και ο διπλός υπολογισμός του ίδιου ζεύγους. Για κάθε ζεύγος αποθηκεύεται η τιμή ομοιότητας στη λίστα `pairwise_sims`. Αν για οποιονδήποτε λόγο η λίστα αυτή μείνει κενή, η συνάρτηση επιστρέφει 0.0 για λόγους ασφαλείας. Τελικά, η `diversity` υπολογίζεται ως $1.0 - \text{np.mean}(\text{pairwise_sims})$, δηλαδή ως ένα μείον ο μέσος όρος των ομοιοτήτων όλων των ζευγών. Επομένως, όσο πιο όμοιες είναι μεταξύ τους οι προτεινόμενες ταινίες, τόσο μικρότερη είναι η τιμή της `diversity`, ενώ όσο περισσότερο διαφέρουν, τόσο μεγαλύτερη είναι η τιμή της.

```

123 # LOAD DATA
124 movies = pd.read_csv(movies_path)
125 tags = pd.read_csv(tags_path)
126 train = pd.read_csv(train_path)
127 test = pd.read_csv(test_path)
128 tmdb = pd.read_csv(tmdb_path)
129
130 for col in ["title", "genres"]:
131     if col in train.columns:
132         train = train.drop(columns=[col])
133     if col in test.columns:
134         test = test.drop(columns=[col])
135
136 print("Movies raw:", movies.shape)
137 print("Tags raw:", tags.shape)
138 print("Train raw:", train.shape)
139 print("Test raw:", test.shape)
140 print("TMDB raw:", tmdb.shape)
141
142 # CANDIDATE CATALOG
143 train_item_ids = set(train["movieId"].unique())
144 test_item_ids = set(test["movieId"].unique())
145
146 print("Unique movies in original movies.csv:", movies["movieId"].nunique())
147 print("Unique movies in TRAIN:", len(train_item_ids))
148 print("Unique movies in TEST:", len(test_item_ids))
149
150 movies = movies[movies["movieId"].isin(train_item_ids)].copy()
151
152 print("Candidate movies for recommendation:", movies["movieId"].nunique())

```

Εικόνα 68: Δημιουργία καταλόγου υποψήφιων ταινιών στο hybrid μοντέλο

Στο επόμενο στάδιο πραγματοποιείται η φόρτωση των βασικών δεδομένων που χρησιμοποιούνται από το hybrid σύστημα προτάσεων. Συγκεκριμένα, διαβάζονται τα αρχεία movies, tags, train, test και tmdb, καθώς η υβριδική προσέγγιση χρειάζεται τόσο τα δεδομένα αξιολογήσεων όσο και τα περιγραφικά μεταδεδομένα των ταινιών. Τα train και test περιέχουν τις αξιολογήσεις των χρηστών και χρησιμοποιούνται αντίστοιχα για την εκπαίδευση και την αξιολόγηση του μοντέλου. Το movies περιλαμβάνει τις βασικές πληροφορίες των ταινιών, όπως movieId, title και genres, το tags περιέχει ετικέτες που έχουν δώσει οι χρήστες, ενώ το tmdb περιλαμβάνει εμπλουτισμένα μεταδεδομένα, όπως keywords, cast, director και άλλα χαρακτηριστικά που θα χρησιμοποιηθούν στο content-based μέρος του hybrid recommender.

Προχωρώντας, ο κώδικας ελέγχει αν οι στήλες title και genres υπάρχουν μέσα στα train και test και, αν υπάρχουν, τις αφαιρεί. Η ενέργεια αυτή γίνεται μόνο στα DataFrames που βρίσκονται στη μνήμη και δεν αλλάζει τα αρχικά αρχεία. Ο λόγος είναι ότι τα train και test πρέπει να περιέχουν κυρίως τις πληροφορίες αξιολόγησης, δηλαδή χρήστη, ταινία και rating, ενώ τα μεταδεδομένα των ταινιών αντλούνται από το movies και το tmdb. Με αυτόν τον τρόπο αποφεύγονται διπλές ή περιττές στήλες και η δομή των δεδομένων παραμένει πιο καθαρή.

Αμέσως μετά, εκτυπώνονται οι αρχικές διαστάσεις των πινάκων με τη χρήση της .shape. Οι εκτυπώσεις αυτές λειτουργούν ως έλεγχος ότι τα δεδομένα φορτώθηκαν σωστά και ότι το μέγεθός τους είναι το αναμενόμενο. Στο hybrid μοντέλο αυτό είναι ιδιαίτερα σημαντικό, επειδή το σύστημα συνδυάζει δεδομένα από πολλές διαφορετικές πηγές. Επομένως, πριν ξεκινήσει η επεξεργασία, χρειάζεται να επιβεβαιωθεί ότι όλα τα απαραίτητα DataFrames έχουν φορτωθεί κανονικά.

Στη συνέχεια ορίζεται το candidate catalog, δηλαδή το σύνολο των ταινιών από το οποίο το σύστημα θα μπορεί τελικά να παράγει προτάσεις. Αρχικά, από το train και το test εξάγονται τα μοναδικά movieId και αποθηκεύονται στα train_item_ids και test_item_ids. Με αυτόν τον τρόπο εντοπίζεται ποιες ταινίες εμφανίζονται πραγματικά στα δύο σύνολα αξιολογήσεων. Είναι σημαντικό να διευκρινιστεί ότι τα train και test δεν αποτελούν διαχωρισμό του ίδιου του πίνακα ταινιών, αλλά διαχωρισμό των αξιολογήσεων. Επομένως, οι ταινίες που συμμετέχουν σε κάθε split προκύπτουν έμμεσα από τα movieId που εμφανίζονται στις αντίστοιχες αξιολογήσεις.

Τέλος, ο πίνακας movies φιλτράρεται ώστε να διατηρηθούν μόνο οι ταινίες που εμφανίζονται στο training set. Αυτό σημαίνει ότι το τελικό candidate catalog περιορίζεται στα αντικείμενα του train. Η επιλογή αυτή είναι συνεπής με τη λογική που ακολουθήθηκε και στα προηγούμενα μοντέλα, καθώς το σύστημα παράγει προτάσεις μόνο για ταινίες που ανήκουν στον χώρο εκπαίδευσης. Στο hybrid μοντέλο αυτό είναι ακόμη πιο σημαντικό, επειδή οι ίδιες υποψήφιες ταινίες πρέπει να μπορούν να χρησιμοποιηθούν τόσο από το collaborative μέρος, μέσω του ALS, όσο και από το content-based μέρος, μέσω της TF-IDF αναπαράστασης. Έτσι, το recommendation space παραμένει κοινό και ελεγχόμενο για τα δύο επιμέρους components του υβριδικού συστήματος.

```

154 # CLEAN MOVIES
155 movies["genres"] = movies["genres"].fillna("").replace("no genres listed", "")
156 movies["genres_text"] = movies["genres"].str.replace("|", " ", regex=False).apply(clean_text)
157
158 movies["title"] = movies["title"].fillna("")
159 movies["title_text"] = movies["title"].apply(clean_text)
160
161 # CLEAN TAGS
162 tags["tag"] = tags["tag"].fillna("").astype(str).apply(clean_text)
163 tags = tags[tags["movieId"].isin(train_item_ids)].copy()
164 tags = tags[tags["tag"].str.len() >= 3]
165 tags = tags[~tags["tag"].str.fullmatch(r"\d+")]
166
167 tag_counts = tags["tag"].value_counts()
168 valid_tags = tag_counts[tag_counts >= 10].index
169 tags = tags[tags["tag"].isin(valid_tags)].copy()
170
171 print("Tags after cleaning:", tags.shape)
172
173 movie_tags = tags.groupby("movieId")["tag"].apply(lambda x: " ".join(x)).reset_index()
174 movie_tags.rename(columns={"tag": "tags_text"}, inplace=True)
175
176 movies = movies.merge(movie_tags, on="movieId", how="left")
177 movies["tags_text"] = movies["tags_text"].fillna("")

```

Εικόνα 69: Καθαρισμός ταινιών και tags στο hybrid μοντέλο

Σε δεύτερη φάση, πραγματοποιείται ο καθαρισμός των βασικών περιγραφικών χαρακτηριστικών των ταινιών και των tags, ώστε τα δεδομένα να είναι κατάλληλα για τη δημιουργία του content-based μέρους του hybrid recommender. Αρχικά, στη στήλη genres αντικαθίστανται οι κενές τιμές με κενό string και η ειδική τιμή (no genres listed) αφαιρείται, καθώς δεν προσφέρει ουσιαστική πληροφορία για το περιεχόμενο της ταινίας. Στη συνέχεια, οι χαρακτήρες | που χωρίζουν τα genres αντικαθίστανται με κενά, ώστε τα επιμέρους είδη να μπορούν να αντιμετωπιστούν ως ξεχωριστές λέξεις από τον TF-IDF vectorizer. Η συνάρτηση clean_text εφαρμόζεται στο αποτέλεσμα, μετατρέποντας το κείμενο σε καθαρή και ομοιόμορφη μορφή. Με παρόμοιο τρόπο καθαρίζεται και η στήλη title: οι κενές τιμές αντικαθίστανται με κενό string και στη συνέχεια δημιουργείται η στήλη title_text, η οποία περιέχει τον καθαρισμένο τίτλο κάθε ταινίας.

Ταυτόχρονα, ακολουθεί ο καθαρισμός των tags. Αρχικά, οι κενές τιμές στη στήλη tag αντικαθίστανται με κενό string, όλες οι τιμές μετατρέπονται σε string και εφαρμόζεται η clean_text, ώστε τα tags να έχουν ενιαία μορφή. Έπειτα, διατηρούνται μόνο τα tags που αντιστοιχούν σε ταινίες του training set, καθώς το τελικό recommendation space έχει ήδη περιοριστεί στα train items. Στη συνέχεια αφαιρούνται tags με μήκος μικρότερο από τρεις

χαρακτήρες, επειδή είναι πιθανό να μη μεταφέρουν ουσιαστική περιγραφική πληροφορία, καθώς και tags που αποτελούνται αποκλειστικά από αριθμούς. Με αυτόν τον τρόπο μειώνεται ο θόρυβος και παραμένουν μόνο πιο χρήσιμες λεκτικές πληροφορίες για τις ταινίες.

Στο πλαίσιο αυτό, ο κώδικας υπολογίζει τη συχνότητα εμφάνισης κάθε tag μέσω της `value_counts()` και κρατά μόνο τα tags που εμφανίζονται τουλάχιστον 10 φορές στο dataset. Η επιλογή αυτή αφαιρεί πολύ σπάνια tags, τα οποία μπορεί να είναι υπερβολικά ειδικά, ανορθόγραφα ή μη αντιπροσωπευτικά, και βοηθά στη δημιουργία πιο σταθερής TF-IDF αναπαράστασης. Μετά το φιλτράρισμα, εκτυπώνεται το τελικό μέγεθος του πίνακα tags, ώστε να υπάρχει έλεγχος του αριθμού των εγγραφών που παρέμειναν μετά τον καθαρισμό.

Το στάδιο αυτό ολοκληρώνεται με τα καθαρισμένα tags να ομαδοποιούνται ανά `movieId`. Για κάθε ταινία, όλα τα tags που της αντιστοιχούν ενώνονται σε ένα ενιαίο κείμενο με την `" ".join(x)`, ώστε να δημιουργηθεί μία συγκεντρωτική περιγραφική στήλη για κάθε ταινία. Η στήλη αυτή μετονομάζεται σε `tags_text` και στη συνέχεια συγχωνεύεται με τον πίνακα `movies`. Με αυτόν τον τρόπο, κάθε ταινία αποκτά μια επιπλέον κειμενική περιγραφή που βασίζεται στα tags των χρηστών. Όπου δεν υπάρχουν διαθέσιμα tags για κάποια ταινία, η τιμή συμπληρώνεται με κενό string. Το αποτέλεσμα αυτού του σταδίου είναι ότι ο πίνακας `movies` περιέχει πλέον καθαρισμένα genres, καθαρισμένο τίτλο και συγκεντρωμένα tags, τα οποία θα αξιοποιηθούν αργότερα στη δημιουργία του `rich_content` και της TF-IDF αναπαράστασης του hybrid μοντέλου.

```

179 # TMDB ENRICHMENT
180 tmdb_cols = [
181     "movieId",
182     "overview",
183     "tagline",
184     "keywords_tmdb",
185     "director",
186     "cast",
187     "production_companies",
188     "collection",
189     "original_title",
190     "genres_tmdb"
191 ]
192 tmdb = tmdb[tmdb_cols].copy()
193 tmdb = tmdb[tmdb["movieId"].isin(train_item_ids)].copy()
194 for col in tmdb_cols[1:]:
195     tmdb[col] = tmdb[col].fillna("").apply(clean_text)
196 movies = movies.merge(tmdb, on="movieId", how="left")
197 for col in [
198     "overview",
199     "tagline",
200     "keywords_tmdb",
201     "director",
202     "cast",
203     "production_companies",
204     "collection",
205     "original_title",
206     "genres_tmdb"
207 ]:
208     movies[col] = movies[col].fillna("")
209 movies["overview"] = movies["overview"].apply(lambda x: truncate_words(x, max_words=60))

```

Εικόνα 70: Ενσωμάτωση μεταδεδομένων TMDB στο hybrid μοντέλο

Η ανάλυση συνεχίζεται με τον εμπλουτισμό των ταινιών με μεταδεδομένα από το TMDb, ώστε το content-based μέρος του hybrid συστήματος να μη βασίζεται μόνο στα βασικά πεδία του MovieLens, αλλά να αξιοποιεί περισσότερη περιγραφική πληροφορία. Αρχικά, στη λίστα tmdb_cols ορίζονται οι στήλες που θα χρησιμοποιηθούν από το αρχείο εμπλουτισμού. Εκτός από το movieId, το οποίο είναι απαραίτητο για τη σύνδεση με τον πίνακα movies, επιλέγονται πεδία όπως overview, tagline, keywords_tmdb, director, cast, production_companies, collection, original_title και genres_tmdb. Τα πεδία αυτά περιγράφουν διαφορετικές πλευρές μιας ταινίας, όπως την υπόθεση, τις λέξεις-κλειδιά, τους βασικούς συντελεστές, τη συλλογή στην οποία ανήκει και τα είδη της σύμφωνα με το TMDb.

Στη συνέχεια, ο πίνακας tmdb περιορίζεται μόνο στις συγκεκριμένες στήλες και κρατά μόνο τις ταινίες που ανήκουν στο training catalog, δηλαδή όσες έχουν movieId μέσα στο

`train_item_ids`. Αυτό γίνεται ώστε ο εμπλουτισμός να είναι ευθυγραμμισμένος με το τελικό `candidate catalog` του μοντέλου και να μη φορτώνονται άσκοπα μεταδεδομένα για ταινίες που δεν μπορούν να προταθούν. Αμέσως μετά, για κάθε στήλη του TMDb εκτός από το `movieId`, οι κενές τιμές αντικαθίστανται με κενό string και εφαρμόζεται η `clean_text`. Έτσι όλα τα κειμενικά πεδία καθαρίζονται με τον ίδιο τρόπο: μετατρέπονται σε πεζά, αφαιρούνται ειδικοί χαρακτήρες και διατηρείται μια ομοιόμορφη μορφή κατάλληλη για TF-IDF επεξεργασία.

Συμπληρωματικά, τα καθαρισμένα TMDb δεδομένα συγχωνεύονται με τον πίνακα `movies` μέσω του κοινού πεδίου `movieId` και με `how="left"`. Αυτό σημαίνει ότι διατηρούνται όλες οι ταινίες που υπάρχουν ήδη στον πίνακα `movies`, ακόμη και αν για κάποιες από αυτές δεν υπάρχουν διαθέσιμα TMDb μεταδεδομένα. Για να αποφευχθούν κενές τιμές μετά τη συγχώνευση, ο κώδικας συμπληρώνει όλες τις σχετικές στήλες με κενό string όπου υπάρχουν NaN τιμές. Με αυτόν τον τρόπο, κάθε ταινία μπορεί να συμμετέχει στη δημιουργία του `rich_content`, ακόμη και αν δεν διαθέτει πλήρη πληροφορία από το TMDb. Συνολικά, το στάδιο αυτό αυξάνει την εκφραστικότητα της περιγραφικής αναπαράστασης των ταινιών, καθώς προσθέτει πληροφορίες που μπορούν να βοηθήσουν το hybrid μοντέλο να εντοπίσει πιο ουσιαστικές ομοιότητες περιεχομένου. Τέλος, στη στήλη `overview` εφαρμόζεται η συνάρτηση `truncate_words` με όριο τις 60 λέξεις. Η ενέργεια αυτή γίνεται επειδή το `overview` αποτελεί συνήθως μεγαλύτερο περιγραφικό κείμενο σε σχέση με τα υπόλοιπα πεδία, όπως τα `genres`, τα `keywords` ή το `cast`. Αν χρησιμοποιούνταν ολόκληρο χωρίς περιορισμό, θα μπορούσε να επηρεάσει δυσανάλογα το τελικό `rich_content` και, κατ' επέκταση, την TF-IDF αναπαράσταση της ταινίας. Με τον περιορισμό στις πρώτες 60 λέξεις διατηρείται η βασική πληροφορία της υπόθεσης, χωρίς το πεδίο αυτό να κυριαρχεί υπερβολικά στο τελικό κειμενικό προφίλ της ταινίας.

```

211 # BUILD RICH CONTENT
212 movies["rich_content"] = (
213     (movies["title_text"] + " ") * 2 +
214     (movies["genres_text"] + " ") * 4 +
215     (movies["genres_tmdb"] + " ") * 4 +
216     (movies["keywords_tmdb"] + " ") * 3 +
217     (movies["director"] + " ") * 3 +
218     (movies["cast"] + " ") * 3 +
219     (movies["collection"] + " ") * 2 +
220     (movies["tags_text"] + " ") * 1 +
221     (movies["tagline"] + " ") * 0 +
222     (movies["overview"] + " ") * 0 +
223     (movies["original_title"] + " ") * 1
224 ).str.strip()
225
226 movies["rich_content"] = movies["rich_content"].replace("", "unknown")

```

Εικόνα 71: Δημιουργία rich content στο hybrid μοντέλο

Στο πλαίσιο αυτό, δημιουργείται η στήλη rich_content, η οποία αποτελεί την τελική ενιαία κειμενική αναπαράσταση κάθε ταινίας για το content-based μέρος του hybrid recommender. Η βασική λογική είναι ότι τα επιμέρους περιγραφικά πεδία της ταινίας, όπως ο τίτλος, τα genres, τα TMDb genres, τα keywords, ο σκηνοθέτης, το cast, η συλλογή και τα tags, ενώνονται σε ένα κοινό κείμενο. Αυτό το ενιαίο κείμενο θα χρησιμοποιηθεί στη συνέχεια από τον TfidfVectorizer, ώστε κάθε ταινία να μετατραπεί σε αριθμητικό διάνυσμα περιεχομένου.

Σημαντικό σημείο εδώ είναι ότι τα πεδία δεν συμμετέχουν όλα με την ίδια βαρύτητα. Ο κώδικας επαναλαμβάνει ορισμένα πεδία περισσότερες φορές, ώστε να αποκτήσουν μεγαλύτερη σημασία στη μετέπειτα TF-IDF αναπαράσταση. Για παράδειγμα, το title_text επαναλαμβάνεται 2 φορές, τα genres_text και genres_tmdb επαναλαμβάνονται 4 φορές, ενώ τα keywords_tmdb, director και cast επαναλαμβάνονται 3 φορές. Με αυτόν τον τρόπο δίνεται μεγαλύτερη έμφαση στα χαρακτηριστικά που θεωρούνται πιο χρήσιμα για την περιγραφή της ταυτότητας μιας ταινίας. Τα genres και τα TMDb genres ενισχύονται περισσότερο, επειδή αποτυπώνουν άμεσα το είδος και τη θεματική κατηγορία της ταινίας. Τα keywords, ο σκηνοθέτης και το cast ενισχύονται επίσης, επειδή μπορούν να συνδέσουν ταινίες με κοινά θεματικά στοιχεία ή κοινά δημιουργικά χαρακτηριστικά.

Αντίθετα, κάποια πεδία συμμετέχουν με μικρότερη ή μηδενική βαρύτητα. Το collection επαναλαμβάνεται 2 φορές, καθώς μπορεί να δείξει ότι μια ταινία ανήκει σε συγκεκριμένη σειρά ή franchise. Τα tags_text και original_title συμμετέχουν μία φορά, ώστε να προσθέσουν επιπλέον πληροφορία χωρίς να κυριαρχήσουν στο τελικό περιεχόμενο. Αντίθετα, τα πεδία

tagline και overview πολλαπλασιάζονται με 0, άρα στην πράξη δεν συμμετέχουν στο τελικό rich_content. Αυτό σημαίνει ότι, παρότι τα δεδομένα αυτά έχουν καθαριστεί προηγουμένως, στη συγκεκριμένη εκδοχή του μοντέλου δεν χρησιμοποιούνται για την TF-IDF αναπαράσταση. Η επιλογή αυτή πιθανότατα έγινε για να μειωθεί ο θόρυβος από πιο ελεύθερα ή μεγάλα κείμενα και να δοθεί μεγαλύτερη έμφαση σε πιο συμπαγή και ελεγχόμενα χαρακτηριστικά, όπως genres, keywords, director και cast.

Στο τελικό κομμάτι, η `.str.strip()` αφαιρεί τυχόν περιττά κενά από την αρχή και το τέλος του τελικού κειμένου. Αμέσως μετά, όσα rich_content έχουν μείνει κενά αντικαθίστανται με την τιμή "unknown". Αυτό γίνεται για λόγους ασφαλείας, ώστε καμία ταινία να μη δοθεί στον TF-IDF vectorizer με εντελώς κενό περιεχόμενο. Συνολικά, το στάδιο αυτό είναι κρίσιμο για το hybrid μοντέλο, καθώς καθορίζει την ποιότητα της content-based αναπαράστασης που θα συνδυαστεί αργότερα με το collaborative score του ALS.

```

228 # TF-IDF
229 tfidf = TfidfVectorizer(
230     stop_words="english",
231     max_features=80000,
232     ngram_range=(1, 2),
233     min_df=7,
234     max_df=0.8,
235     sublinear_tf=True
236 )
237
238 tfidf_matrix = tfidf.fit_transform(movies["rich_content"])
239 print("TF-IDF matrix:", tfidf_matrix.shape)
240
241 movie_id_to_idx = pd.Series(np.arange(len(movies)), index=movies["movieId"]).to_dict()
242
243 # align train/test to candidate catalog
244 train = train[train["movieId"].isin(movie_id_to_idx)].copy()
245 test = test[test["movieId"].isin(movie_id_to_idx)].copy()
246
247 print("Train after alignment:", train.shape)
248 print("Test after alignment:", test.shape)
249 print("Final candidate movie count:", len(movie_id_to_idx))
250
251 # USER / ITEM MAPS FOR ALS
252 user_ids = train["userId"].unique()
253 item_ids = train["movieId"].unique()
254
255 user_map = {uid: i for i, uid in enumerate(user_ids)}
256 item_map = {iid: i for i, iid in enumerate(item_ids)}
257 item_map_rev = {v: k for k, v in item_map.items()}

```

Εικόνα 72: TF-IDF αναπαράσταση και αντιστοίχιση IDs στο hybrid μοντέλο

Το επόμενο βήμα αφορά την δημιουργία της TF-IDF αναπαράστασης των ταινιών, η οποία αποτελεί τη βάση του content-based μέρους του hybrid recommender. Ο TfidfVectorizer εφαρμόζεται στη στήλη rich_content, δηλαδή στο ενιαίο κείμενο που δημιουργήθηκε προηγουμένως για κάθε ταινία. Με την επιλογή stop_words="english" αφαιρούνται συχνές αγγλικές λέξεις που συνήθως δεν προσφέρουν ουσιαστική πληροφορία, όπως άρθρα και βοηθητικές λέξεις. Η παράμετρος max_features=80000 περιορίζει το λεξιλόγιο στα 80.000 σημαντικότερα χαρακτηριστικά, ώστε η αναπαράσταση να παραμένει πλούσια αλλά υπολογιστικά διαχειρίσιμη. Με το ngram_range=(1, 2) ο vectorizer λαμβάνει υπόψη τόσο μεμονωμένες λέξεις όσο και ζεύγη διαδοχικών λέξεων, κάτι που βοηθά στην καλύτερη αποτύπωση σύνθετων όρων, όπως ονόματα, θεματικές εκφράσεις ή συνδυασμοί keywords. Οι παράμετροι min_df=7 και max_df=0.8 λειτουργούν ως φίλτρα συχνότητας: το πρώτο αφαιρεί

πολύ σπάνιους όρους που εμφανίζονται σε λιγότερες από 7 ταινίες, ενώ το δεύτερο αφαιρεί υπερβολικά κοινούς όρους που εμφανίζονται σε πάνω από το 80% των ταινιών. Τέλος, το `sublinear_tf=True` εφαρμόζει λογαριθμική κλιμάκωση στη συχνότητα των όρων, ώστε η επανάληψη μιας λέξης να αυξάνει τη σημασία της με πιο ήπιο τρόπο και όχι γραμμικά.

Στη συνέχεια, η εντολή `tfidf.fit_transform(movies["rich_content"])` εκπαιδεύει τον TF-IDF vectorizer πάνω στο `rich_content` των ταινιών και ταυτόχρονα μετατρέπει κάθε ταινία σε αριθμητικό διάνυσμα. Το αποτέλεσμα αποθηκεύεται στη μεταβλητή `tfidf_matrix`, όπου κάθε γραμμή αντιστοιχεί σε μία ταινία και κάθε στήλη σε ένα λεκτικό χαρακτηριστικό που προέκυψε από το λεξιλόγιο του TF-IDF. Η εκτύπωση του σχήματος της μήτρας επιτρέπει να ελεγχθεί πόσες ταινίες και πόσα τελικά χαρακτηριστικά περιλαμβάνονται στην αναπαράσταση. Αμέσως μετά, δημιουργείται το λεξικό `movie_id_to_idx`, το οποίο αντιστοιχίζει κάθε πραγματικό `movieId` στη θέση της αντίστοιχης ταινίας μέσα στον πίνακα `movies` και στη `tfidf_matrix`. Η αντιστοίχιση αυτή είναι απαραίτητη, επειδή στα δεδομένα χρησιμοποιούνται τα πραγματικά `movieId`, ενώ για την πρόσβαση στις γραμμές της TF-IDF μήτρας χρειάζονται αριθμητικοί δείκτες.

Έπειτα, τα σύνολα `train` και `test` ευθυγραμμίζονται με το τελικό `candidate catalog`, κρατώντας μόνο τις εγγραφές των οποίων το `movieId` υπάρχει στο `movie_id_to_idx`. Με αυτόν τον τρόπο διασφαλίζεται ότι κάθε αξιολόγηση που παραμένει στο `train` και στο `test` αντιστοιχεί σε ταινία για την οποία υπάρχει διαθέσιμη TF-IDF αναπαράσταση. Οι εκτυπώσεις που ακολουθούν εμφανίζουν τις διαστάσεις των `train` και `test` μετά την ευθυγράμμιση, καθώς και το τελικό πλήθος των υποψήφιων ταινιών. Αυτό είναι σημαντικό στο `hybrid` μοντέλο, επειδή οι ίδιες ταινίες πρέπει να είναι διαθέσιμες τόσο στο `content-based` μέρος όσο και στο `collaborative` μέρος.

Στο τελευταίο κομμάτι του αποσπάσματος δημιουργούνται οι αντιστοιχίσεις χρηστών και ταινιών που χρειάζονται για το ALS. Από το `train` εξάγονται τα μοναδικά `userId` και `movieId`, τα οποία αποθηκεύονται στις μεταβλητές `user_ids` και `item_ids`. Στη συνέχεια δημιουργείται το `user_map`, που αντιστοιχίζει κάθε χρήστη σε έναν εσωτερικό ακέραιο δείκτη, και το `item_map`, που κάνει το ίδιο για κάθε ταινία. Αυτοί οι δείκτες είναι απαραίτητοι επειδή η μήτρα αλληλεπιδράσεων του ALS δεν χρησιμοποιεί απευθείας τα πραγματικά IDs του MovieLens, αλλά αριθμητικές θέσεις γραμμών και στηλών. Τέλος, δημιουργείται και το `item_map_rev`, το οποίο επιτρέπει την αντίστροφη αντιστοίχιση από τον εσωτερικό δείκτη

πίσω στο πραγματικό movieId. Αυτό θα χρειαστεί αργότερα, όταν το ALS θα επιστρέψει προτάσεις με βάση εσωτερικά item indices και ο κώδικας θα πρέπει να τις μετατρέψει ξανά σε πραγματικές ταινίες.

```

259 # INTERACTION MATRIX
260 rows = train["userId"].map(user_map)
261 cols = train["movieId"].map(item_map)
262 data = train["rating"]
263
264 interaction_matrix = coo_matrix(
265     arg1=(data, (rows, cols)),
266     shape=(len(user_ids), len(item_ids))
267 )
268
269 train_csr = interaction_matrix.tocsr()
270
271 print("Interaction matrix shape:", interaction_matrix.shape)
272
273 # LOAD OR TRAIN ALS
274 if model_path.exists():
275     with open(model_path, "rb") as f:
276         als = pickle.load(f)
277     print("ALS model loaded from disk")
278 else:
279     als = AlternatingLeastSquares(
280         factors=50,
281         regularization=0.1,
282         iterations=20,
283         random_state=42
284     )
285     als.fit(train_csr)
286     print("ALS model trained")
287
288     with open(model_path, "wb") as f:
289         pickle.dump(als, f)
290     print("ALS model saved to disk")

```

Εικόνα 73: Πίνακας αλληλεπιδράσεων και εκπαίδευση ALS στο hybrid μοντέλο

Προχωράμε στην δημιουργία της μήτρας αλληλεπιδράσεων χρηστών-ταινιών, η οποία αποτελεί τη βάση του collaborative μέρους του hybrid recommender. Αρχικά, τα πραγματικά `userId` και `movieId` του `train` μετατρέπονται στους αντίστοιχους εσωτερικούς αριθμητικούς δείκτες μέσω των λεξικών `user_map` και `item_map`. Οι δείκτες των χρηστών αποθηκεύονται στη μεταβλητή `rows`, οι δείκτες των ταινιών στη μεταβλητή `cols`, ενώ οι πραγματικές βαθμολογίες αποθηκεύονται στη μεταβλητή `data`. Με αυτόν τον τρόπο, κάθε εγγραφή του `training set` μετατρέπεται σε τριάδα της μορφής: χρήστης, ταινία, βαθμολογία.

Σε πρώτο πλάνο, με τη χρήση της `coo_matrix`, οι τριάδες αυτές μετατρέπονται σε `sparse` μήτρα αλληλεπιδράσεων. Οι γραμμές της μήτρας αντιστοιχούν στους χρήστες, οι στήλες αντιστοιχούν στις ταινίες και τα μη κενά κελιά περιέχουν τις βαθμολογίες που έχουν δοθεί. Η `sparse` αναπαράσταση είναι απαραίτητη, επειδή σε ένα recommender system οι περισσότεροι χρήστες έχουν αξιολογήσει μόνο ένα πολύ μικρό μέρος των διαθέσιμων ταινιών. Άρα, αν η μήτρα αποθηκευόταν ως κανονικός `dense` πίνακας, θα περιείχε τεράστιο αριθμό κενών θέσεων και θα απαιτούσε υπερβολική μνήμη. Με τη `sparse` μορφή αποθηκεύονται μόνο οι πραγματικές αλληλεπιδράσεις που υπάρχουν στο `dataset`.

Αμέσως μετά, η `interaction_matrix` μετατρέπεται σε μορφή CSR με την εντολή `train_csr = interaction_matrix.tocsr()`. Η CSR μορφή, δηλαδή Compressed Sparse Row, είναι ένας τρόπος αποθήκευσης `sparse` πινάκων που οργανώνει τα μη μηδενικά στοιχεία ανά γραμμή. Αυτό σημαίνει ότι αντί να αποθηκεύεται ολόκληρος ο πίνακας χρηστών-ταινιών, μαζί με όλα τα κενά κελιά, αποθηκεύονται μόνο οι πραγματικές αλληλεπιδράσεις που υπάρχουν. Η συγκεκριμένη μορφή είναι ιδιαίτερα αποδοτική όταν χρειάζεται γρήγορη πρόσβαση στις γραμμές του πίνακα, δηλαδή στις αλληλεπιδράσεις κάθε χρήστη. Αυτό είναι σημαντικό για το ALS, επειδή κατά την εκπαίδευση και την παραγωγή προτάσεων το μοντέλο χρειάζεται συχνά να ανακτά τα `ratings` ή τις αλληλεπιδράσεις ενός συγκεκριμένου χρήστη. Επομένως, η μετατροπή σε CSR δεν αλλάζει το περιεχόμενο της μήτρας, αλλά βελτιώνει τον τρόπο με τον οποίο αυτή αποθηκεύεται και χρησιμοποιείται υπολογιστικά. Η εκτύπωση `Interaction matrix shape` δείχνει τις τελικές διαστάσεις της μήτρας, δηλαδή πόσοι χρήστες και πόσες ταινίες συμμετέχουν στο collaborative μέρος του μοντέλου. Με άλλα λόγια, σε αυτό το σημείο έχει δημιουργηθεί ο πίνακας πάνω στον οποίο το ALS θα μάθει τα μοτίβα αξιολόγησης των χρηστών.

Στο δεύτερο μέρος του αποσπάσματος ο κώδικας ελέγχει αν υπάρχει ήδη αποθηκευμένο εκπαιδευμένο μοντέλο ALS στη διαδρομή `model_path`. Αν το αρχείο υπάρχει, τότε ανοίγεται σε λειτουργία ανάγνωσης δυαδικών δεδομένων και φορτώνεται με τη χρήση της `pickle.load`. Σε αυτή την περίπτωση, το μοντέλο δεν εκπαιδεύεται ξανά, αλλά επαναχρησιμοποιείται από προηγούμενη εκτέλεση. Αυτό είναι ιδιαίτερα χρήσιμο σε τόσο μεγάλο dataset, επειδή η εκπαίδευση του ALS μπορεί να απαιτεί σημαντικό χρόνο και υπολογιστικούς πόρους.

Αν δεν υπάρχει ήδη αποθηκευμένο μοντέλο, τότε δημιουργείται νέο αντικείμενο `AlternatingLeastSquares`. Η παράμετρος `factors=50` ορίζει το πλήθος των latent παραγόντων που θα χρησιμοποιηθούν για την αναπαράσταση χρηστών και ταινιών. Με απλά λόγια, κάθε χρήστης και κάθε ταινία μετατρέπεται από το ALS σε ένα διάνυσμα 50 διαστάσεων. Οι διαστάσεις αυτές δεν αντιστοιχούν άμεσα σε εμφανή χαρακτηριστικά, όπως `genre` ή `σκηνοθέτης`, αλλά σε κρυφά μοτίβα που μαθαίνει το μοντέλο από τις αξιολογήσεις. Για παράδειγμα, κάποιοι latent παράγοντες μπορεί να συλλαμβάνουν έμμεσα προτιμήσεις προς συγκεκριμένα είδη ταινιών, στυλ, δημοφιλία ή συνδυασμούς χαρακτηριστικών, χωρίς όμως να έχουν άμεση ονομασία. Η παράμετρος `regularization=0.1` βοηθά στον περιορισμό της υπερπροσαρμογής, λειτουργώντας ως μηχανισμός ελέγχου, αποτρέποντας τα latent factors από το να πάρουν υπερβολικά μεγάλες ή ακραίες τιμές. Έτσι, το μοντέλο γίνεται πιο σταθερό και έχει καλύτερες πιθανότητες να αποδώσει σωστά και στο test set. Παράλληλα, η `iterations=20` καθορίζει πόσες επαναλήψεις θα πραγματοποιηθούν κατά την εκπαίδευση και η `random_state=42` χρησιμοποιείται για πιο σταθερά και αναπαραγώγιμα αποτελέσματα. Στη συνέχεια, το μοντέλο εκπαιδεύεται πάνω στη μήτρα `train_csr` με την `als.fit(train_csr)`. Μετά την εκπαίδευση, το μοντέλο αποθηκεύεται στον δίσκο με `pickle.dump`, ώστε σε επόμενες εκτελέσεις να μπορεί να φορτωθεί απευθείας χωρίς να επαναληφθεί η διαδικασία εκπαίδευσης. Έτσι, το συγκεκριμένο στάδιο ολοκληρώνει τη δημιουργία ή φόρτωση του collaborative component που θα χρησιμοποιηθεί αργότερα στον υβριδικό συνδυασμό.

```

292 # FAST USER LOOKUP
293 user_train_dict = {uid: df.copy() for uid, df in train.groupby("userId")}
294
295 # CONTENT USER PROFILE
296 def build_content_user_profile(user_id): 1 usage
297     user_train = user_train_dict.get(user_id)
298     if user_train is None or user_train.empty:
299         return None, None
300
301     user_likes = user_train[user_train["rating"] >= PROFILE_THRESHOLD].copy()
302     if user_likes.empty:
303         return None, None
304
305     user_likes = user_likes[user_likes["movieId"].isin(movie_id_to_idx)].copy()
306     if user_likes.empty:
307         return None, None
308
309     item_indices = user_likes["movieId"].map(movie_id_to_idx).to_numpy()
310     ratings_array = user_likes["rating"].to_numpy()
311
312     weights = get_rating_weights(ratings_array, mode=WEIGHT_MODE)
313
314     item_matrix_local = tfidf_matrix[item_indices]
315     weighted_matrix = item_matrix_local.multiply(weights)
316
317     weight_sum = weights.sum()
318     if weight_sum == 0:
319         return None, None
320
321     user_profile = np.asarray(weighted_matrix.sum(axis=0) / weight_sum).reshape(1, -1)
322     seen_movie_ids = set(user_train["movieId"].unique())
323     return user_profile, seen_movie_ids

```

Εικόνα 74: Δημιουργία προφίλ χρήστη στο hybrid μοντέλο

Σύμφωνα με το παραπάνω απόσπασμα δημιουργείται αρχικά το λεξικό `user_train_dict`, το οποίο λειτουργεί ως γρήγορος μηχανισμός αναζήτησης του ιστορικού κάθε χρήστη. Συγκεκριμένα, το `train` ομαδοποιείται με βάση το `userId` μέσω της `groupby("userId")` και για κάθε χρήστη αποθηκεύεται ένα ξεχωριστό `DataFrame` με τις αξιολογήσεις του. Με αυτόν τον τρόπο, όταν στη συνέχεια χρειάζεται να ανακτηθεί το ιστορικό ενός συγκεκριμένου χρήστη, ο κώδικας δεν χρειάζεται να φιλτράρει κάθε φορά ολόκληρο το `train dataset`, αλλά μπορεί να το βρει άμεσα από το λεξικό. Αυτό είναι ιδιαίτερα σημαντικό στο `MovieLens 32M`, καθώς το `training set` περιέχει εκατομμύρια εγγραφές και η επαναλαμβανόμενη αναζήτηση πάνω σε ολόκληρο τον πίνακα θα ήταν πιο αργή.

Κατόπιν ορίζεται η συνάρτηση `build_content_user_profile`, η οποία δημιουργεί το content-based προφίλ ενός συγκεκριμένου χρήστη. Το προφίλ αυτό θα χρησιμοποιηθεί αργότερα στο hybrid recommender, ώστε το τελικό σύστημα να λαμβάνει υπόψη όχι μόνο το collaborative score του ALS, αλλά και το περιεχόμενο των ταινιών που έχει αξιολογήσει θετικά ο χρήστης. Αρχικά, η συνάρτηση αναζητά το ιστορικό του χρήστη μέσα στο `user_train_dict`. Αν δεν υπάρχει ιστορικό ή αν το αντίστοιχο DataFrame είναι κενό, επιστρέφει `None, None`, καθώς δεν υπάρχουν δεδομένα πάνω στα οποία μπορεί να βασιστεί η δημιουργία προφίλ.

Έπειτα, από το ιστορικό του χρήστη κρατούνται μόνο οι ταινίες που έχουν βαθμολογία μεγαλύτερη ή ίση από το `PROFILE_THRESHOLD`, δηλαδή από το 4.0 και πάνω. Οι ταινίες αυτές θεωρούνται θετικές αξιολογήσεις και χρησιμοποιούνται για να περιγράψουν τις προτιμήσεις του χρήστη. Αν ο χρήστης δεν έχει καμία τέτοια θετική αξιολόγηση, η συνάρτηση επιστρέφει ξανά `None, None`, επειδή δεν υπάρχει επαρκής θετική πληροφορία για να δημιουργηθεί content profile. Στη συνέχεια, οι θετικά αξιολογημένες ταινίες φιλτράρονται ώστε να διατηρηθούν μόνο όσες υπάρχουν στο `movie_id_to_idx`, δηλαδή μόνο όσες έχουν διαθέσιμη TF-IDF αναπαράσταση στο τελικό candidate catalog. Αν μετά από αυτό το φιλτράρισμα δεν μείνει καμία ταινία, η συνάρτηση επίσης σταματά.

Αφού εντοπιστούν οι κατάλληλες θετικά αξιολογημένες ταινίες, τα `movieId` τους μετατρέπονται στα αντίστοιχα indices της `tfidf_matrix` μέσω του `movie_id_to_idx`. Παράλληλα, οι βαθμολογίες τους αποθηκεύονται στο `ratings_array`. Στη συνέχεια, οι βαθμολογίες αυτές μετατρέπονται σε βάρη μέσω της `get_rating_weights`, με βάση το `WEIGHT_MODE`. Επειδή στο συγκεκριμένο πείραμα έχει οριστεί `WEIGHT_MODE = "discrete_strong"`, οι υψηλότερες βαθμολογίες αποκτούν μεγαλύτερη βαρύτητα στο τελικό προφίλ του χρήστη. Έτσι, μια ταινία που έχει αξιολογηθεί με 5.0 επηρεάζει περισσότερο το προφίλ από μια ταινία που έχει αξιολογηθεί απλώς θετικά με 4.0.

Παράλληλα, από τη `tfidf_matrix` ανακτώνται τα TF-IDF διανύσματα των θετικά αξιολογημένων ταινιών του χρήστη και αποθηκεύονται στη μεταβλητή `item_matrix_local`. Αυτά τα διανύσματα πολλαπλασιάζονται με τα αντίστοιχα βάρη των ratings μέσω της `multiply(weights)`. Με αυτόν τον τρόπο, οι ταινίες που άρεσαν περισσότερο στον χρήστη συμβάλλουν πιο έντονα στη διαμόρφωση του τελικού content profile. Αφού υπολογιστεί το άθροισμα των βαρών, γίνεται έλεγχος αν αυτό είναι μηδέν, ώστε να αποφευχθεί πιθανή διαίρεση με μηδέν.

Το τελικό `user_profile` προκύπτει ως ο σταθμισμένος μέσος όρος των TF-IDF διανυσμάτων των θετικά αξιολογημένων ταινιών. Δηλαδή, η συνάρτηση αθροίζει τα σταθμισμένα διανύσματα και τα διαιρεί με το συνολικό άθροισμα των βαρών. Το αποτέλεσμα είναι ένα ενιαίο διάνυσμα που συνοψίζει τις προτιμήσεις του χρήστη ως προς το περιεχόμενο των ταινιών που του άρεσαν. Η μορφή του μετατρέπεται σε πίνακα μίας γραμμής με `reshape(1, -1)`, ώστε να μπορεί αργότερα να συγκριθεί με την TF-IDF μήτρα όλων των υποψήφιων ταινιών μέσω `cosine similarity`.

Τέλος, η συνάρτηση δημιουργεί το σύνολο `seen_movie_ids`, το οποίο περιέχει όλες τις ταινίες που έχει ήδη αξιολογήσει ο χρήστης στο `training set`. Αυτό το σύνολο είναι απαραίτητο στο στάδιο παραγωγής προτάσεων, ώστε το σύστημα να μην προτείνει ξανά ταινίες που ο χρήστης έχει ήδη δει ή αξιολογήσει. Η συνάρτηση επιστρέφει δύο στοιχεία: το `user_profile`, δηλαδή το content-based προφίλ του χρήστη, και το `seen_movie_ids`, δηλαδή το σύνολο των ήδη γνωστών ταινιών του.

```
325 # HYBRID RECOMMENDER
326 def recommend_hybrid(user_id, top_k=10, alpha=0.70): 2 usages
327     if user_id not in user_map:
328         return pd.DataFrame(columns=[
329             "movieId", "title", "genres",
330             "collab_score", "content_score", "final_score"
331         ])
332
333     content_profile, seen_movie_ids = build_content_user_profile(user_id)
334     if seen_movie_ids is None:
335         seen_movie_ids = set()
336
337     # unseen candidates over the SAME catalog
338     candidate_mask = ~movies["movieId"].isin(seen_movie_ids)
339     candidate_movie_ids = movies.loc[candidate_mask, "movieId"].to_numpy()
340
341     if len(candidate_movie_ids) == 0:
342         return pd.DataFrame(columns=[
343             "movieId", "title", "genres",
344             "collab_score", "content_score", "final_score"
345         ])
346
347     candidate_item_indices = np.array([item_map[mid] for mid in candidate_movie_ids], dtype=np.int32)
```

Εικόνα 75: Αρχικοποίηση υβριδικών συστάσεων

Στο συγκεκριμένο τμήμα ξεκινά η βασική συνάρτηση παραγωγής προτάσεων του υβριδικού μοντέλου, `recommend_hybrid`. Η συνάρτηση δέχεται ως είσοδο το `user_id`, δηλαδή τον χρήστη για τον οποίο θα παραχθούν προτάσεις, το `top_k`, που καθορίζει πόσες προτάσεις θα επιστραφούν, και το `alpha`, το οποίο καθορίζει το βάρος του collaborative μέρους στον

τελικό υβριδικό υπολογισμό. Στη συγκεκριμένη υλοποίηση, η προεπιλεγμένη τιμή του alpha είναι 0.70, δηλαδή το τελικό score θα βασιστεί περισσότερο στο collaborative component και λιγότερο στο content-based component.

Αρχικά, η συνάρτηση ελέγχει αν ο χρήστης υπάρχει στο `user_map`. Ο έλεγχος αυτός είναι απαραίτητος, επειδή το collaborative μέρος του hybrid recommender βασίζεται στο ALS και το ALS μπορεί να παράγει score μόνο για χρήστες που υπάρχουν στο training set και έχουν αντιστοιχιστεί σε εσωτερικό αριθμητικό δείκτη. Αν ο χρήστης δεν υπάρχει στο `user_map`, η συνάρτηση επιστρέφει ένα κενό DataFrame με τις στήλες που θα είχε κανονικά η έξοδος: `movieId`, `title`, `genres`, `collab_score`, `content_score` και `final_score`. Με αυτόν τον τρόπο διατηρείται σταθερή η δομή της εξόδου, ακόμη και όταν δεν μπορούν να παραχθούν προτάσεις.

Ταυτόχρονα, καλείται η συνάρτηση `build_content_user_profile(user_id)`, η οποία επιστρέφει το content-based προφίλ του χρήστη και το σύνολο των ταινιών που έχει ήδη αξιολογήσει στο training set. Το `content_profile` θα χρησιμοποιηθεί αργότερα για τον υπολογισμό του content-based score, ενώ το `seen_movie_ids` χρησιμοποιείται για να αποκλειστούν από τις προτάσεις οι ταινίες που ο χρήστης έχει ήδη δει ή αξιολογήσει. Αν για κάποιον λόγο το `seen_movie_ids` είναι None, μετατρέπεται σε κενό σύνολο, ώστε να μπορεί να χρησιμοποιηθεί με ασφάλεια στο επόμενο φιλτράρισμα.

Επίσης, δημιουργείται το σύνολο των υποψήφιων ταινιών προς πρόταση. Η μεταβλητή `candidate_mask` κρατά τις ταινίες των οποίων το `movieId` δεν βρίσκεται μέσα στο `seen_movie_ids`. Με άλλα λόγια, από το κοινό candidate catalog αφαιρούνται οι ταινίες που είναι ήδη γνωστές στον χρήστη. Στη συνέχεια, με τη χρήση του `mask`, εξάγονται τα `movieId` των υποψήφιων ταινιών και αποθηκεύονται στη μεταβλητή `candidate_movie_ids`. Το σημαντικό εδώ είναι ότι οι υποψήφιες ταινίες προέρχονται από το ίδιο catalog που χρησιμοποιείται τόσο από το content-based όσο και από το collaborative μέρος. Έτσι, το υβριδικό μοντέλο υπολογίζει και συνδυάζει scores πάνω στον ίδιο χώρο αντικειμένων.

Αν μετά τον αποκλεισμό των ήδη αξιολογημένων ταινιών δεν απομείνει καμία υποψήφια ταινία, η συνάρτηση επιστρέφει ξανά κενό DataFrame με τις προβλεπόμενες στήλες. Τέλος, τα `candidate_movie_ids` μετατρέπονται στους αντίστοιχους εσωτερικούς δείκτες του ALS μέσω του `item_map` και αποθηκεύονται στη μεταβλητή `candidate_item_indices`. Η μετατροπή αυτή είναι απαραίτητη, επειδή το collaborative μέρος του μοντέλου δεν δουλεύει

με τα πραγματικά movieId, αλλά με τους εσωτερικούς αριθμητικούς δείκτες που χρησιμοποιούνται στα als.item_factors. Έτσι, μέχρι αυτό το σημείο η συνάρτηση έχει εντοπίσει τον χρήστη, έχει δημιουργήσει το content προφίλ του, έχει αφαιρέσει τις ήδη γνωστές ταινίες και έχει προετοιμάσει τις υποψήφιες ταινίες ώστε να μπορούν να βαθμολογηθούν τόσο από το collaborative όσο και από το content-based component.

```

349 # ----- collaborative scores -----
350 user_idx = user_map[user_id]
351 user_factor = als.user_factors[user_idx]
352 item_factors = als.item_factors[candidate_item_indices]
353 collab_scores_raw = item_factors @ user_factor
354 collab_scores = minmax_id(collab_scores_raw)
355
356 # ----- content scores -----
357 if content_profile is not None:
358     all_content_scores = cosine_similarity(content_profile, tfidf_matrix).flatten()
359     candidate_content_indices = np.array([movie_id_to_idx[mid] for mid in candidate_movie_ids], dtype=np.int32)
360     content_scores_raw = all_content_scores[candidate_content_indices]
361     content_scores = minmax_id(content_scores_raw)
362 else:
363     content_scores = np.zeros(len(candidate_movie_ids), dtype=np.float32)
364
365 # ----- hybrid -----
366 final_scores = alpha * collab_scores + (1.0 - alpha) * content_scores
367 recs = pd.DataFrame({
368     "movieId": candidate_movie_ids,
369     "collab_score": collab_scores,
370     "content_score": content_scores,
371     "final_score": final_scores
372 })
373 recs = recs.merge(movies[["movieId", "title", "genres"]], on="movieId", how="left")
374 recs = recs.sort_values(by="final_score", ascending=False).head(top_k).reset_index(drop=True)
375 return recs[[
376     "movieId", "title", "genres",
377     "collab_score", "content_score", "final_score"
378 ]]

```

Εικόνα 76: Συνδυασμός scores στο hybrid μοντέλο

Στο παρακάτω στάδιο της συνάρτησης υπολογίζονται τα δύο επιμέρους scores που θα χρησιμοποιηθούν από το hybrid recommender: το collaborative score και το content-based score. Αρχικά, στο collaborative μέρος, το πραγματικό user_id μετατρέπεται στον εσωτερικό δείκτη του ALS μέσω του user_map. Με βάση αυτόν τον δείκτη, ανακτάται από το als.user_factors το λανθάνον διάνυσμα του χρήστη, δηλαδή το user_factor. Αντίστοιχα, από το als.item_factors ανακτώνται τα λανθάνοντα διανύσματα όλων των υποψήφιων ταινιών, μέσω των candidate_item_indices. Τα διανύσματα αυτά δεν βασίζονται σε περιγραφικά χαρακτηριστικά, όπως genres ή cast, αλλά στα μοτίβα αξιολογήσεων που έχει μάθει το ALS από το training set. Στη συνέχεια, το collab_scores_raw = item_factors @ user_factor

υπολογίζει το γινόμενο ανάμεσα στο διάνυσμα του χρήστη και στα διανύσματα των υποψήφιων ταινιών. Με απλά λόγια, για κάθε υποψήφια ταινία υπολογίζεται πόσο ταιριάζει στον συγκεκριμένο χρήστη με βάση το collaborative μοντέλο. Επειδή όμως τα raw ALS scores δεν είναι απαραίτητα στην ίδια κλίμακα με τα content scores, εφαρμόζεται η `minmax_1d`, ώστε τα collaborative scores να μετατραπούν σε τιμές από 0 έως 1.

Προχωράμε με τον υπολογισμό του content-based score. Αν υπάρχει διαθέσιμο `content_profile` για τον χρήστη, τότε η συνάρτηση `cosine_similarity(content_profile, tfidf_matrix)` συγκρίνει το προφίλ του χρήστη με όλα τα TF-IDF διανύσματα των ταινιών. Το αποτέλεσμα είναι ένας πίνακας ομοιοτήτων, όπου κάθε τιμή δείχνει πόσο κοντά βρίσκεται μια ταινία στις προτιμήσεις περιεχομένου του χρήστη. Με το `.flatten()` ο πίνακας μετατρέπεται σε μονοδιάστατη μορφή, ώστε να είναι πιο εύκολη η επεξεργασία του. Στη συνέχεια, τα `candidate_movie_ids` μετατρέπονται στα αντίστοιχα `indices` της TF-IDF μήτρας μέσω του `movie_id_to_idx`, και από το σύνολο των content scores κρατούνται μόνο οι τιμές που αντιστοιχούν στις υποψήφιες ταινίες. Όπως και στο collaborative μέρος, τα raw content scores κανονικοποιούνται με `minmax_1d`, ώστε να βρίσκονται στην ίδια κλίμακα από 0 έως 1. Αν δεν υπάρχει διαθέσιμο content profile, τότε το content score όλων των υποψήφιων ταινιών ορίζεται ίσο με 0, ώστε το σύστημα να μπορεί να συνεχίσει να λειτουργεί βασιζόμενο μόνο στο collaborative component.

Αφού υπολογιστούν και κανονικοποιηθούν τα δύο επιμέρους scores, δημιουργείται το τελικό hybrid score. Στη συγκεκριμένη υλοποίηση, το alpha έχει τιμή 0.70, επομένως το τελικό score δίνει μεγαλύτερη βαρύτητα στο collaborative μέρος και μικρότερη στο content-based μέρος. Πρακτικά, αυτό σημαίνει ότι το μοντέλο βασίζεται κυρίως στα μοτίβα αξιολογήσεων που έχει μάθει το ALS, αλλά ταυτόχρονα ενσωματώνει και την ομοιότητα περιεχομένου ανάμεσα στο προφίλ του χρήστη και στις υποψήφιες ταινίες. Έτσι, το τελικό αποτέλεσμα δεν προκύπτει αποκλειστικά από τη συμπεριφορά άλλων χρηστών ούτε αποκλειστικά από τα περιγραφικά χαρακτηριστικά των ταινιών, αλλά από συνδυασμό των δύο.

Συνοψίζοντας, τα αποτελέσματα οργανώνονται σε ένα DataFrame `recs`, το οποίο περιλαμβάνει για κάθε υποψήφια ταινία το `movieId`, το `collab_score`, το `content_score` και το τελικό `final_score`. Στη συνέχεια, το DataFrame συγχωνεύεται με τον πίνακα `movies`, ώστε να προστεθούν ο τίτλος και τα genres κάθε ταινίας. Αφού προστεθούν τα αναγνώσιμα μεταδεδομένα, οι ταινίες ταξινομούνται κατά φθίνουσα σειρά με βάση το `final_score`, ώστε οι

πιο κατάλληλες προτάσεις να εμφανιστούν πρώτες. Με την `.head(top_k)` κρατούνται μόνο οι πρώτες `top_k` ταινίες και με το `reset_index(drop=True)` ανανεώνεται η αρίθμηση των γραμμών. Η συνάρτηση επιστρέφει τελικά έναν καθαρό πίνακα με τις στήλες `movieId`, `title`, `genres`, `collab_score`, `content_score` και `final_score`, ώστε να είναι εμφανές όχι μόνο ποιες ταινίες προτείνονται, αλλά και πώς διαμορφώθηκε η τελική υβριδική βαθμολογία τους.

```

380 # EVALUATION
381 def evaluate_model(sample_size=300, top_k=10, eval_threshold=3.5, alpha=0.70):
382     precision_scores = []
383     recall_scores = []
384     map_scores = []
385     ndcg_scores_list = []
386     hitrate_scores = []
387     diversity_scores = []
388
389     unique_test_users = test["userId"].drop_duplicates()
390     sample_users = unique_test_users.sample(
391         min(sample_size, len(unique_test_users)),
392         random_state=42
393     )
394
395     evaluated_users = 0
396     for user_id in sample_users:
397         if user_id not in user_map:
398             continue
399
400         relevant_items = set(
401             test[
402                 (test["userId"] == user_id) &
403                 (test["rating"] >= eval_threshold)
404             ]["movieId"].unique()
405         )
406
407         if len(relevant_items) == 0:
408             continue

```

Εικόνα 77: Αξιολόγηση hybrid συστήματος συστάσεων

Στο στάδιο της αξιολόγησης ορίζεται η συνάρτηση `evaluate_model`, η οποία χρησιμοποιείται για να υπολογίσει τη συνολική απόδοση του hybrid recommender σε ένα δείγμα χρηστών. Η συνάρτηση δέχεται ως παραμέτρους το `sample_size`, δηλαδή το πλήθος των χρηστών που θα χρησιμοποιηθούν στην αξιολόγηση, το `top_k`, δηλαδή τον αριθμό των προτάσεων που θα εξεταστούν για κάθε χρήστη, το `eval_threshold`, που καθορίζει από ποια

βαθμολογία και πάνω μια ταινία θεωρείται *relevant*, και το *alpha*, το οποίο περνά στη συνάρτηση *recommend_hybrid* ώστε οι προτάσεις να παραχθούν με το ίδιο βάρος *collaborative* και *content-based* συνιστώσας που έχει οριστεί στο μοντέλο. Στην αρχή της συνάρτησης δημιουργούνται ξεχωριστές λίστες για κάθε μετρική αξιολόγησης: *precision_scores*, *recall_scores*, *map_scores*, *ndcg_scores_list*, *hitrate_scores* και *diversity_scores*. Σε αυτές τις λίστες θα αποθηκεύεται η τιμή κάθε μετρικής για κάθε χρήστη που θα αξιολογηθεί επιτυχώς. Με αυτόν τον τρόπο, στο τέλος της διαδικασίας μπορεί να υπολογιστεί ο μέσος όρος κάθε μετρικής για το σύνολο των χρηστών του δείγματος.

Ομοίως, από το *test* εξάγονται οι μοναδικοί χρήστες με τη χρήση της *drop_duplicates()*. Από αυτούς επιλέγεται τυχαίο δείγμα χρηστών με τη συνάρτηση *sample*. Το μέγεθος του δείγματος είναι το μικρότερο ανάμεσα στο *sample_size* και στο πλήθος των διαθέσιμων χρηστών, ώστε να μην υπάρξει σφάλμα αν οι διαθέσιμοι χρήστες είναι λιγότεροι από το ζητούμενο δείγμα. Η παράμετρος *random_state=42* εξασφαλίζει ότι η ίδια δειγματοληψία μπορεί να αναπαραχθεί σε επόμενες εκτελέσεις, κάτι που είναι σημαντικό για τη συνέπεια των πειραμάτων.

Έπειτα αρχικοποιείται η μεταβλητή *evaluated_users*, η οποία μετρά πόσοι χρήστες αξιολογήθηκαν τελικά. Αυτό είναι απαραίτητο, επειδή δεν είναι βέβαιο ότι όλοι οι χρήστες του αρχικού δείγματος θα συμμετάσχουν στην τελική αξιολόγηση. Για κάθε χρήστη του δείγματος, ο κώδικας ελέγχει πρώτα αν υπάρχει στο *user_map*. Αν δεν υπάρχει, παραλείπεται, επειδή το *hybrid* μοντέλο χρειάζεται τον χρήστη στο *collaborative space* του ALS για να μπορέσει να υπολογίσει *collaborative score*.

Στη συνέχεια ορίζονται τα *relevant_items* του χρήστη. Ο κώδικας αναζητά στο *test* τις ταινίες του συγκεκριμένου χρήστη που έχουν βαθμολογία μεγαλύτερη ή ίση από το *eval_threshold*. Αυτές οι ταινίες θεωρούνται οι πραγματικά σχετικές ταινίες με βάση τις οποίες θα συγκριθούν οι προτάσεις του συστήματος. Αν ο χρήστης δεν έχει κανένα *relevant item* στο *test set*, τότε παραλείπεται με *continue*, επειδή δεν υπάρχει βάση σύγκρισης για να υπολογιστούν οι μετρικές αξιολόγησης. Συνεπώς, μέχρι αυτό το σημείο η συνάρτηση έχει προετοιμάσει το δείγμα αξιολόγησης και έχει καθορίσει ποιοι χρήστες μπορούν πράγματι να αξιολογηθούν.

```

410     recs = recommend_hybrid(user_id, top_k=top_k, alpha=alpha)
411     if recs.empty():
412         continue
413
414     recommended_items = recs["movieId"].tolist()
415     recommended_scores = recs["final_score"].tolist()
416
417     precision_scores.append(precision_at_k(recommended_items, relevant_items, top_k))
418     recall_scores.append(recall_at_k(recommended_items, relevant_items, top_k))
419     map_scores.append(average_precision_at_k(recommended_items, relevant_items, top_k))
420     ndcg_scores_list.append(ndcg_at_k(recommended_items, recommended_scores, relevant_items, top_k))
421     hitrate_scores.append(hit_rate_at_k(recommended_items, relevant_items, top_k))
422     diversity_scores.append(intra_list_diversity(recommended_items))
423
424     evaluated_users += 1
425
426     return {
427         "Users evaluated": evaluated_users,
428         f"Precision@{top_k}": np.mean(precision_scores) if precision_scores else 0.0,
429         f"Recall@{top_k}": np.mean(recall_scores) if recall_scores else 0.0,
430         f"MAP@{top_k}": np.mean(map_scores) if map_scores else 0.0,
431         f"NDCG@{top_k}": np.mean(ndcg_scores_list) if ndcg_scores_list else 0.0,
432         f"HitRate@{top_k}": np.mean(hitrate_scores) if hitrate_scores else 0.0,
433         "Diversity": np.mean(diversity_scores) if diversity_scores else 0.0
434     }

```

Εικόνα 78: Υπολογισμός μετρικών αξιολόγησης hybrid μοντέλου

Στο δεύτερο μέρος της αξιολόγησης, για κάθε χρήστη που έχει περάσει τους προηγούμενους ελέγχους, καλείται η συνάρτηση `recommend_hybrid`. Η συνάρτηση αυτή παράγει τις top-k προτάσεις του hybrid μοντέλου, χρησιμοποιώντας την τιμή `alpha` που έχει οριστεί για τον συνδυασμό collaborative και content-based score. Αν το αποτέλεσμα είναι κενό, δηλαδή αν δεν παραχθούν έγκυρες προτάσεις για τον συγκεκριμένο χρήστη, τότε ο χρήστης παραλείπεται και η διαδικασία συνεχίζει με τον επόμενο.

Αν υπάρχουν προτάσεις, ο κώδικας εξάγει δύο βασικά στοιχεία από το αποτέλεσμα. Αρχικά, η στήλη `movieId` μετατρέπεται στη λίστα `recommended_items`, η οποία περιέχει τις ταινίες που πρότείνει το σύστημα. Στη συνέχεια, η στήλη `final_score` μετατρέπεται στη λίστα `recommended_scores`, η οποία περιέχει το τελικό υβριδικό score κάθε πρότασης. Τα `recommended_items` χρησιμοποιούνται για τον υπολογισμό των Precision, Recall, MAP, HitRate και Diversity, ενώ τα `recommended_scores` χρησιμοποιούνται επιπλέον στη NDCG, επειδή η συγκεκριμένη μετρική λαμβάνει υπόψη και τη σειρά/βαθμολόγηση της κατάταξης.

Κάπου εδώ υπολογίζονται και οι μετρικές αξιολόγησης για τον συγκεκριμένο χρήστη. Η `precision_at_k` μετρά πόσες από τις top-k προτάσεις είναι relevant, η `recall_at_k` μετρά πόσα

από τα relevant αντικείμενα του χρήστη ανακτήθηκαν μέσα στις top-k προτάσεις, ενώ η $average_precision_at_k$ λαμβάνει υπόψη και τις θέσεις στις οποίες εμφανίζονται τα relevant αντικείμενα. Η $ndcg_at_k$ αξιολογεί την ποιότητα της κατάταξης με βάση τα $final_score$, η $hit_rate_at_k$ ελέγχει αν υπάρχει έστω μία relevant ταινία μέσα στις top-k προτάσεις και η $intra_list_diversity$ υπολογίζει την εσωτερική ποικιλία της λίστας προτάσεων. Κάθε τιμή προστίθεται στην αντίστοιχη λίστα, ώστε στο τέλος να μπορεί να υπολογιστεί ο μέσος όρος της μετρικής για όλους τους αξιολογημένους χρήστες. Αφού ολοκληρωθεί επιτυχώς η αξιολόγηση του χρήστη, η μεταβλητή `evaluated_users` αυξάνεται κατά ένα.

Κλείνοντας, η συνάρτηση επιστρέφει ένα λεξικό με τα συνολικά αποτελέσματα της αξιολόγησης. Το πεδίο "Users evaluated" δείχνει πόσοι χρήστες αξιολογήθηκαν τελικά, ενώ οι υπόλοιπες τιμές αντιστοιχούν στους μέσους όρους των μετρικών $Precision@K$, $Recall@K$, $MAP@K$, $NDCG@K$, $HitRate@K$ και $Diversity$. Αν κάποια λίστα μετρικών είναι κενή, επιστρέφεται η τιμή 0.0, ώστε να αποφευχθούν σφάλματα σε περιπτώσεις όπου δεν αξιολογήθηκε κανένας χρήστης. Με αυτόν τον τρόπο, η συνάρτηση παρέχει μια συγκεντρωτική εικόνα της απόδοσης του hybrid recommender στο επιλεγμένο δείγμα χρηστών.

```

436 # MAIN
437 if __name__ == "__main__":
438     random_user = np.random.choice(train["userId"].unique())
439     print(f"\nTop positively rated movies already seen by user {random_user}:")
440     user_seen = user_train_dict.get(random_user, pd.DataFrame()).copy()
441     if not user_seen.empty:
442         user_seen = user_seen[user_seen["rating"] >= PROFILE_THRESHOLD].copy()
443         user_seen = user_seen.merge(
444             movies[["movieId", "title", "genres"]],
445             on="movieId",
446             how="left"
447         )
448         user_seen = user_seen.sort_values(by=["rating"], ascending=False).head(15)
449         print(user_seen[["movieId", "title", "rating", "genres"]].reset_index(drop=True))
450     else:
451         print(f"No train history found for user {random_user}")
452
453     print(f"\nTop-{TOP_K} Hybrid recommendations for user {random_user}:")
454     print(recommend_hybrid(random_user, top_k=TOP_K, alpha=ALPHA))
455     print("\nEvaluating hybrid recommender...")
456     results = evaluate_model(
457         sample_size=EVAL_SAMPLE_USERS,
458         top_k=TOP_K,
459         eval_threshold=EVAL_THRESHOLD,
460         alpha=ALPHA
461     )
462     print("\nEvaluation Results:")
463     for metric, value in results.items():
464         if isinstance(value, float):
465             print(f"{metric}: {value:.4f}")
466         else:
467             print(f"{metric}: {value}")

```

Εικόνα 79: Εκτέλεση και αξιολόγηση hybrid μοντέλου

Στο τελικό τμήμα του κώδικα ορίζεται η κύρια ροή εκτέλεσης του hybrid recommender. Η συνθήκη `if __name__ == "__main__":` εξασφαλίζει ότι το συγκεκριμένο μέρος θα εκτελεστεί μόνο όταν το αρχείο τρέχει ως κύριο πρόγραμμα και όχι όταν εισάγεται ως module σε κάποιο άλλο αρχείο. Αρχικά, επιλέγεται τυχαία ένας χρήστης από το training set με την `np.random.choice(train["userId"].unique())`. Ο χρήστης αυτός χρησιμοποιείται για να παρουσιαστεί ένα ενδεικτικό παράδειγμα λειτουργίας του συστήματος, πριν πραγματοποιηθεί η συνολική αξιολόγηση.

Ο κώδικας εμφανίζει τις ταινίες που έχει ήδη αξιολογήσει θετικά ο συγκεκριμένος χρήστης. Το ιστορικό του ανακτάται από το `user_train_dict` και, αν υπάρχει, φιλτράρεται ώστε να κρατηθούν μόνο οι ταινίες με βαθμολογία μεγαλύτερη ή ίση από το

PROFILE_THRESHOLD. Έπειτα, οι εγγραφές αυτές συγχωνεύονται με τον πίνακα movies, ώστε να προστεθούν ο τίτλος και τα genres κάθε ταινίας. Οι ταινίες ταξινομούνται με βάση τη βαθμολογία σε φθίνουσα σειρά και εμφανίζονται οι πρώτες 15. Με αυτόν τον τρόπο, πριν παρουσιαστούν οι προτάσεις, δίνεται μια εικόνα των προτιμήσεων του χρήστη πάνω στις οποίες βασίζεται το content-based μέρος του hybrid μοντέλου. Αν δεν υπάρχει διαθέσιμο ιστορικό για τον χρήστη, εμφανίζεται σχετικό μήνυμα.

Αμέσως μετά, καλείται η recommend_hybrid για τον ίδιο χρήστη και εμφανίζονται οι top-K υβριδικές προτάσεις. Οι προτάσεις αυτές προκύπτουν από τον συνδυασμό του collaborative score του ALS και του content-based score που βασίζεται στο TF-IDF προφίλ του χρήστη. Η τιμή ALPHA περνά στη συνάρτηση ώστε να χρησιμοποιηθεί το προκαθορισμένο βάρος του collaborative component στον τελικό υπολογισμό. Έτσι, το output δείχνει όχι μόνο ποιες ταινίες προτείνει το υβριδικό σύστημα, αλλά και πώς συμπεριφέρεται το μοντέλο για έναν συγκεκριμένο χρήστη με πραγματικό ιστορικό αξιολογήσεων.

Τέλος, ο κώδικας περνά στη συνολική αξιολόγηση του hybrid recommender. Η συνάρτηση evaluate_model καλείται με τις παραμέτρους EVAL_SAMPLE_USERS, TOP_K, EVAL_THRESHOLD και ALPHA, ώστε η αξιολόγηση να γίνει με τις ίδιες ρυθμίσεις που έχουν οριστεί στην αρχή του προγράμματος. Τα αποτελέσματα αποθηκεύονται στο λεξικό results και στη συνέχεια εκτυπώνονται ένα προς ένα. Αν η τιμή μιας μετρικής είναι δεκαδικός αριθμός, εμφανίζεται με τέσσερα δεκαδικά ψηφία, ενώ διαφορετικά εκτυπώνεται όπως είναι. Με αυτόν τον τρόπο, το κύριο μέρος του προγράμματος συνδυάζει ποιοτική παρουσίαση ενός παραδείγματος προτάσεων για τυχαίο χρήστη και ποσοτική αξιολόγηση της συνολικής απόδοσης του hybrid μοντέλου.

4.3.2 Παρουσίαση αποτελεσμάτων αρχείου hybrid.py

```
Movies raw: (87585, 3)
Tags raw: (2000072, 4)
Train raw: (25474164, 6)
Test raw: (6368541, 6)
TMDB raw: (31478, 21)
Unique movies in original movies.csv: 87585
Unique movies in TRAIN: 31961
Unique movies in TEST: 31445
Candidate movies for recommendation: 31961
Tags after cleaning: (1650095, 4)
TF-IDF matrix: (31961, 65162)
Train after alignment: (25474164, 6)
Test after alignment: (6368541, 6)
Final candidate movie count: 31961
Interaction matrix shape: (200948, 31961)
ALS model loaded from disk

Top positively rated movies already seen by user 151229:
```

	movieId	title	rating	genres
0	7147	Big Fish (2003)	4.5	Drama Fantasy Romance
1	69406	Proposal, The (2009)	4.5	Comedy Romance
2	81562	127 Hours (2010)	4.5	Adventure Drama Thriller
3	70599	Time Traveler's Wife, The (2009)	4.0	Drama Romance Sci-Fi
4	4155	Sweet November (2001)	4.0	Drama Romance
5	39427	Stay (2005)	4.0	Thriller
6	2607	Get Real (1998)	4.0	Drama Romance
7	58559	Dark Knight, The (2008)	4.0	Action Crime Drama IMAX
8	1682	Truman Show, The (1998)	4.0	Comedy Drama Sci-Fi

Εικόνα 80: Προεπεξεργασία και ιστορικό χρήστη στο hybrid μοντέλο

Στην εκτέλεση του υβριδικού μοντέλου αρχικά επιβεβαιώνεται η σωστή φόρτωση των δεδομένων και η δημιουργία του κοινού candidate catalog. Το αρχικό αρχείο movies.csv περιλαμβάνει 87.585 ταινίες, ενώ τα αρχεία train και test περιλαμβάνουν 25.474.164 και 6.368.541 αξιολογήσεις αντίστοιχα. Από το σύνολο των ταινιών, 31.961 εμφανίζονται στο training set και 31.445 στο test set, με αποτέλεσμα το τελικό candidate catalog να περιορίζεται στις 31.961 ταινίες του train. Η επιλογή αυτή ακολουθεί την ίδια λογική με τα προηγούμενα μοντέλα, ώστε οι προτάσεις να παράγονται μόνο από αντικείμενα που είναι γνωστά στο

σύστημα κατά την εκπαίδευση. Μετά τον καθαρισμό των tags παραμένουν 1.650.095 εγγραφές, οι οποίες χρησιμοποιούνται για τον εμπλουτισμό της περιγραφικής πληροφορίας των ταινιών. Η τελική TF-IDF μήτρα έχει διαστάσεις (31961, 65162), γεγονός που σημαίνει ότι οι 31.961 υποψήφιες ταινίες αναπαρίστανται μέσω 65.162 λεκτικών χαρακτηριστικών. Παράλληλα, η μήτρα αλληλεπιδράσεων του ALS έχει διαστάσεις (200948, 31961), δηλαδή περιλαμβάνει 200.948 χρήστες και 31.961 ταινίες. Τέλος, το μήνυμα ALS model loaded from disk δείχνει ότι το collaborative μέρος του υβριδικού συστήματος δεν εκπαιδεύτηκε ξανά στη συγκεκριμένη εκτέλεση, αλλά φορτώθηκε από ήδη αποθηκευμένο μοντέλο.

Προχωρώντας, εμφανίζεται ένα ενδεικτικό παράδειγμα για τον χρήστη 151229. Αρχικά παρουσιάζονται οι ταινίες που έχει ήδη αξιολογήσει θετικά στο training set, με βαθμολογίες από 4.0 και πάνω. Το ιστορικό του συγκεκριμένου χρήστη δείχνει προτίμηση κυρίως σε ταινίες Drama, Romance, Fantasy, Thriller και Sci-Fi, ενώ εμφανίζονται και ορισμένες πιο έντονες επιλογές όπως το *The Dark Knight*. Η εμφάνιση αυτού του ιστορικού βοηθά στην ποιοτική κατανόηση των προτάσεων που ακολουθούν, καθώς δείχνει το προφίλ προτιμήσεων πάνω στο οποίο βασίζεται το content-based μέρος του hybrid μοντέλου.

```

Top-10 Hybrid recommendations for user 151229:
  movieId    ...  final_score
0      79132    ...    0.948558
1       7361    ...    0.945573
2       2959    ...    0.827345
3       4878    ...    0.803197
4      48780    ...    0.799190
5       1704    ...    0.788346
6        318    ...    0.782035
7      74458    ...    0.760098
8      69757    ...    0.747854
9       1246    ...    0.728800

[10 rows x 6 columns]

Evaluating hybrid recommender...

Evaluation Results:
Users evaluated: 293
Precision@10: 0.3123
Recall@10: 0.2565
MAP@10: 0.2781
NDCG@10: 0.6713
HitRate@10: 0.8942
Diversity: 0.9094

```

Εικόνα 81: Συστάσεις και αξιολόγηση hybrid μοντέλου

Αμέσως μετά παρουσιάζονται οι top-10 υβριδικές προτάσεις για τον ίδιο χρήστη. Κάθε πρόταση συνοδεύεται από το τελικό `final_score`, το οποίο προκύπτει από τον συνδυασμό του collaborative score και του content-based score. Στη συγκεκριμένη υλοποίηση, το βάρος του collaborative μέρους είναι $\text{ALPHA} = 0.70$, επομένως το τελικό αποτέλεσμα επηρεάζεται κυρίως από τα μοτίβα αξιολόγησης που έχει μάθει το ALS, αλλά ενσωματώνει και την ομοιότητα περιεχομένου μέσω TF-IDF. Οι προτάσεις που εμφανίζονται αποτελούν την τελική ταξινομημένη λίστα του hybrid recommender για τον συγκεκριμένο χρήστη, με τις υψηλότερες τιμές `final_score` να τοποθετούνται στις πρώτες θέσεις.

Κλείνοντας, εμφανίζονται συνοπτικά τα αποτελέσματα αξιολόγησης του hybrid μοντέλου. Από τους 300 χρήστες που επιλέχθηκαν αρχικά για αξιολόγηση, τελικά αξιολογήθηκαν 293. Για το συγκεκριμένο πείραμα προέκυψαν οι τιμές $\text{Precision@10} = 0.3123$, $\text{Recall@10} = 0.2565$, $\text{MAP@10} = 0.2781$, $\text{NDCG@10} = 0.6713$, $\text{HitRate@10} = 0.8942$ και $\text{Diversity} = 0.9094$. Οι τιμές αυτές δίνουν μια πρώτη συνολική εικόνα της απόδοσης του

υβριδικού συστήματος ως προς τη συνάφεια, την ποιότητα κατάταξης, την ύπαρξη τουλάχιστον μίας επιτυχημένης πρότασης και την ποικιλία των τελικών λιστών.

4.3.3 Αξιολόγηση Μοντέλου

Στο τελευταίο στάδιο της εκτέλεσης πραγματοποιείται η αξιολόγηση του υβριδικού συστήματος προτάσεων σε τυχαίο δείγμα χρηστών από το test set. Για τη συγκεκριμένη εκτέλεση επιλέχθηκαν αρχικά 300 χρήστες, από τους οποίους τελικά αξιολογήθηκαν 293. Οι υπόλοιποι χρήστες παραλείφθηκαν είτε επειδή δεν διέθεταν relevant αντικείμενα στο test set είτε επειδή δεν ήταν δυνατή η παραγωγή έγκυρων προτάσεων για αυτούς. Ως relevant αντικείμενα θεωρούνται οι ταινίες του test set που έχουν αξιολογηθεί με βαθμολογία μεγαλύτερη ή ίση από το προκαθορισμένο κατώφλι $EVAL_THRESHOLD = 3.5$. Για κάθε χρήστη το μοντέλο παράγει τις 10 κορυφαίες προτάσεις και στη συνέχεια οι προτάσεις αυτές συγκρίνονται με τα relevant αντικείμενα του χρήστη. Τα τελικά αποτελέσματα της αξιολόγησης είναι:

- $Precision@10 = 0.3123$
- $Recall@10 = 0.2565$
- $MAP@10 = 0.2781$
- $NDCG@10 = 0.6713$
- $HitRate@10 = 0.8942$
- $Diversity = 0.9094$

Η τιμή $Precision@10 = 0.3123$ δείχνει ότι, κατά μέσο όρο, περίπου 3,12 από τις 10 προτεινόμενες ταινίες είναι relevant για τον χρήστη. Η μετρική αυτή αποτυπώνει πόσο στοχευμένη είναι η τελική λίστα προτάσεων. Όσο υψηλότερο είναι το Precision, τόσο μεγαλύτερο ποσοστό των διαθέσιμων θέσεων στο top-10 καταλαμβάνεται από ταινίες που συμφωνούν με τα relevant αντικείμενα του test set. Στην περίπτωση του hybrid μοντέλου, η συγκεκριμένη τιμή δείχνει ότι η λίστα προτάσεων περιλαμβάνει αρκετές σωστές επιλογές και ότι ο συνδυασμός collaborative και content-based πληροφορίας βοηθά στην παραγωγή πιο στοχευμένων αποτελεσμάτων.

Η τιμή $Recall@10 = 0.2565$ δείχνει ότι το σύστημα ανακτά περίπου το 25,65% των relevant ταινιών του χρήστη μέσα στις πρώτες δέκα προτάσεις. Σε αντίθεση με το Precision,

που εξετάζει πόσες από τις προτάσεις ήταν σωστές, το Recall εξετάζει πόσο καλά καλύπτεται το σύνολο των σχετικών αντικειμένων του χρήστη. Η συγκεκριμένη τιμή δείχνει ότι το μοντέλο καταφέρνει να εντοπίσει ένα σημαντικό μέρος των relevant επιλογών, παρότι περιορίζεται σε λίστα μόλις 10 προτάσεων. Αυτό είναι ιδιαίτερα σημαντικό, καθώς σε ένα πραγματικό recommender system ο χρήστης συνήθως βλέπει μικρό αριθμό προτάσεων και επομένως η ανάκτηση σχετικών αντικειμένων μέσα στις πρώτες θέσεις έχει μεγάλη πρακτική αξία.

Η τιμή $MAP@10 = 0.2781$ δείχνει την ποιότητα της κατάταξης λαμβάνοντας υπόψη τις θέσεις στις οποίες εμφανίζονται οι relevant ταινίες. Η MAP δεν εξετάζει απλώς αν το σύστημα βρήκε σχετικές ταινίες, αλλά επιβραβεύει περισσότερο τις περιπτώσεις όπου αυτές εμφανίζονται νωρίς στη λίστα. Επομένως, η συγκεκριμένη τιμή δείχνει ότι το hybrid μοντέλο δεν εντοπίζει απλώς relevant αντικείμενα, αλλά σε αρκετές περιπτώσεις τα τοποθετεί και σε καλές θέσεις μέσα στο top-10. Αυτό είναι σημαντικό, επειδή οι πρώτες θέσεις της λίστας έχουν μεγαλύτερη πιθανότητα να γίνουν αντιληπτές από τον χρήστη.

Αντίστοιχα, η τιμή $NDCG@10 = 0.6713$ δείχνει ότι η σειρά των προτάσεων βρίσκεται σε αρκετά καλό επίπεδο σε σχέση με μια ιδανική κατάταξη. Η NDCG δίνει μεγαλύτερη βαρύτητα στα relevant αντικείμενα που εμφανίζονται στις πρώτες θέσεις και μειώνει τη σημασία των relevant αντικειμένων που βρίσκονται χαμηλότερα στη λίστα. Η τιμή αυτή είναι αρκετά υψηλή και δείχνει ότι το σύστημα καταφέρνει να τοποθετεί τις πιο χρήσιμες προτάσεις σχετικά ψηλά. Επομένως, το hybrid μοντέλο εμφανίζει ικανοποιητική συμπεριφορά όχι μόνο ως προς την επιλογή των ταινιών, αλλά και ως προς τη διάταξή τους.

Η τιμή $HitRate@10 = 0.8942$ σημαίνει ότι στο 89,42% των αξιολογημένων χρηστών το σύστημα κατάφερε να εμφανίσει τουλάχιστον μία relevant ταινία μέσα στις πρώτες δέκα προτάσεις. Η συγκεκριμένη μετρική είναι ιδιαίτερα χρήσιμη για την πρακτική αξιολόγηση ενός συστήματος σύστασης, επειδή δείχνει αν ο χρήστης λαμβάνει έστω μία πρόταση που μπορεί να θεωρηθεί χρήσιμη. Η υψηλή αυτή τιμή δείχνει ότι το hybrid recommender σπάνια αποτυγχάνει πλήρως να εντοπίσει σχετικό περιεχόμενο για τον χρήστη. Ωστόσο, η HitRate δεν δείχνει πόσες relevant ταινίες υπάρχουν στη δεκάδα ούτε σε ποιες θέσεις βρίσκονται· για αυτό πρέπει να ερμηνεύεται μαζί με τις υπόλοιπες μετρικές.

Τέλος, η τιμή $Diversity = 0.9094$ δείχνει ότι οι τελικές λίστες προτάσεων παρουσιάζουν υψηλό βαθμό εσωτερικής ποικιλίας. Στη συγκεκριμένη υβριδική υλοποίηση, η Diversity

υπολογίζεται στον χώρο περιεχομένου, δηλαδή με βάση τα TF-IDF διανύσματα των προτεινόμενων ταινιών. Επομένως, η τιμή αυτή δείχνει ότι οι προτεινόμενες ταινίες δεν είναι υπερβολικά όμοιες μεταξύ τους ως προς τα περιγραφικά τους χαρακτηριστικά. Με άλλα λόγια, το μοντέλο δεν περιορίζεται σε μια πολύ στενή ομάδα παρόμοιων ταινιών, αλλά παράγει λίστες που περιλαμβάνουν ταινίες με σημαντική εσωτερική διαφοροποίηση.

Συνολικά, τα αποτελέσματα δείχνουν ότι το hybrid μοντέλο παρουσιάζει πολύ καλή συμπεριφορά ως προς τη συνάφεια, την κατάταξη και την ποικιλία των προτάσεων. Το Precision@10 και το Recall@10 δείχνουν ότι το σύστημα καταφέρνει να εντοπίζει relevant ταινίες μέσα σε περιορισμένη λίστα δέκα προτάσεων. Το MAP@10 και το NDCG@10 δείχνουν ότι οι relevant ταινίες τείνουν να εμφανίζονται σε καλές θέσεις, ενώ το HitRate@10 επιβεβαιώνει ότι στη μεγάλη πλειονότητα των χρηστών υπάρχει τουλάχιστον μία επιτυχημένη πρόταση. Παράλληλα, η υψηλή τιμή της Diversity δείχνει ότι η αποτελεσματικότητα αυτή δεν επιτυγχάνεται εις βάρος της ποικιλίας, καθώς οι τελικές λίστες παραμένουν αρκετά διαφοροποιημένες ως προς το περιεχόμενο. Συνεπώς, το υβριδικό σύστημα φαίνεται να αξιοποιεί αποτελεσματικά τόσο τα συλλογικά μοτίβα αξιολόγησης του ALS όσο και την περιγραφική πληροφορία των ταινιών από το content-based μέρος.

5. Συμπεράσματα

Στο παρόν κεφάλαιο παρουσιάζονται τα βασικά συμπεράσματα που προέκυψαν από την ανάπτυξη, εφαρμογή και αξιολόγηση των τριών συστημάτων σύστασης που υλοποιήθηκαν στην εργασία. Συγκεκριμένα, εξετάστηκαν τρεις διαφορετικές προσεγγίσεις: ένα content-based μοντέλο, το οποίο βασίζεται στα περιγραφικά χαρακτηριστικά των ταινιών και στο ιστορικό προτιμήσεων κάθε χρήστη, ένα collaborative filtering μοντέλο βασισμένο σε ALS matrix factorization, το οποίο αξιοποιεί τα μοτίβα αξιολογήσεων χρηστών-ταινιών, και ένα hybrid μοντέλο, το οποίο συνδυάζει τις δύο προηγούμενες προσεγγίσεις μέσω σταθμισμένου αθροίσματος των επιμέρους scores. Με αυτόν τον τρόπο, η εργασία δεν περιορίστηκε στην εξέταση μίας μόνο τεχνικής, αλλά επιχείρησε να συγκρίνει τρεις διαφορετικούς τρόπους παραγωγής προτάσεων, αναδεικνύοντας τα πλεονεκτήματα και τους περιορισμούς κάθε προσέγγισης.

Η αξιολόγηση πραγματοποιήθηκε στο MovieLens 32M dataset, το οποίο αποτελεί ένα μεγάλο και απαιτητικό σύνολο δεδομένων για συστήματα σύστασης ταινιών. Για λόγους συνέπειας και συγκρισιμότητας, και τα τρία μοντέλα αξιολογήθηκαν με κοινό train-test split και κοινό candidate catalog. Αυτό σημαίνει ότι τα μοντέλα εκπαιδεύτηκαν και αξιολογήθηκαν στο ίδιο πειραματικό πλαίσιο, ενώ οι τελικές προτάσεις παράγονταν από τον ίδιο χώρο υποψήφιων ταινιών. Η επιλογή αυτή είναι σημαντική, καθώς μειώνει τον κίνδυνο άνισης σύγκρισης και επιτρέπει την πιο αξιόπιστη αποτίμηση της πραγματικής συμπεριφοράς κάθε μοντέλου.

Η σύγκριση βασίστηκε σε ranking-oriented μετρικές, όπως Precision@10, Recall@10, MAP@10, NDCG@10 και HitRate@10. Οι μετρικές αυτές κρίθηκαν πιο κατάλληλες για το αντικείμενο της εργασίας, καθώς το ζητούμενο δεν ήταν η ακριβής πρόβλεψη της αριθμητικής βαθμολογίας που θα έδινε ένας χρήστης σε μια ταινία, αλλά η παραγωγή ποιοτικών top-10 λιστών προτάσεων. Επομένως, η αξιολόγηση εστίασε στο αν οι σχετικές ταινίες εμφανίζονται μέσα στις πρώτες δέκα προτάσεις, στο πόσες από αυτές είναι πράγματι relevant και στο αν τοποθετούνται σε υψηλές θέσεις της κατάταξης. Παράλληλα, χρησιμοποιήθηκε και η μετρική Diversity, ώστε να εξεταστεί η εσωτερική ποικιλία των προτεινόμενων ταινιών. Η προσθήκη αυτής της μετρικής ήταν σημαντική, καθώς ένα σύστημα σύστασης δεν αξιολογείται μόνο από το αν προτείνει σχετικές ταινίες, αλλά και από το αν οι τελικές λίστες του παρουσιάζουν επαρκή διαφοροποίηση και δεν περιορίζονται σε υπερβολικά όμοια αντικείμενα.

5.1 Σύγκριση των τριών μοντέλων

Τα αποτελέσματα της αξιολόγησης των τριών μοντέλων παρουσιάζονται συνοπτικά στον παρακάτω πίνακα:

Μοντέλο	Precision@10	Recall@10	MAP@10	NDCG@10	HitRate@10	Diversity
Content-Based	0.1911	0.1251	0.1470	0.5094	0.7099	0.8677
Collaborative	0.2949	0.2373	0.2463	0.6438	0.8703	0.5875
Hybrid	0.3123	0.2565	0.2781	0.6713	0.8942	0.9094

Πίνακας 1: Σύγκριση μετρήσεων των 3 μοντέλων

Από τα αποτελέσματα φαίνεται ότι το content-based μοντέλο παρουσιάζει τη χαμηλότερη απόδοση στις βασικές μετρικές συνάφειας και κατάταξης. Συγκεκριμένα, το Precision@10 έχει τιμή 0.1911, γεγονός που σημαίνει ότι κατά μέσο όρο περίπου 1,91 από τις 10 προτεινόμενες ταινίες ήταν relevant. Το Recall@10 ανέρχεται σε 0.1251, δείχνοντας ότι το μοντέλο ανακτά σχετικά μικρό μέρος των relevant αντικειμένων του χρήστη. Αντίστοιχα, οι τιμές MAP@10 = 0.1470 και NDCG@10 = 0.5094 δείχνουν ότι, παρότι το μοντέλο μπορεί να εντοπίσει ορισμένες σχετικές ταινίες, δεν τις κατατάσσει πάντα στις υψηλότερες θέσεις της λίστας. Η τιμή HitRate@10 = 0.7099 δείχνει ότι περίπου στο 71% των χρηστών υπήρξε τουλάχιστον μία relevant πρόταση μέσα στο top-10.

Η συμπεριφορά αυτή είναι αναμενόμενη για ένα content-based μοντέλο, καθώς βασίζεται αποκλειστικά στα χαρακτηριστικά περιεχομένου των ταινιών και στο ιστορικό του κάθε χρήστη. Το μοντέλο μπορεί να εντοπίζει ταινίες με παρόμοια περιγραφικά χαρακτηριστικά με εκείνες που έχει αξιολογήσει θετικά ο χρήστης, όμως δεν αξιοποιεί τη συλλογική συμπεριφορά άλλων χρηστών. Επομένως, δυσκολεύεται να προτείνει ταινίες που δεν είναι εμφανώς κοντά στο υπάρχον προφίλ του χρήστη, ακόμη και αν αυτές θα μπορούσαν να είναι σχετικές με βάση τα μοτίβα αξιολόγησης άλλων χρηστών. Παρ' όλα αυτά, το content-based μοντέλο παρουσιάζει υψηλή Diversity, με τιμή 0.8677, γεγονός που δείχνει ότι οι προτεινόμενες ταινίες παρουσιάζουν σημαντική εσωτερική ποικιλία ως προς το περιεχόμενό τους.

Το collaborative μοντέλο ALS παρουσιάζει σαφώς καλύτερη απόδοση στις περισσότερες μετρικές. Το Precision@10 αυξάνεται σε 0.2949, κάτι που σημαίνει ότι κατά μέσο όρο περίπου 2,95 από τις 10 προτάσεις είναι relevant. Το Recall@10 φτάνει στο

0.2373, δείχνοντας ότι το μοντέλο ανακτά μεγαλύτερο μέρος των σχετικών ταινιών σε σχέση με το content-based. Επίσης, οι τιμές $MAP@10 = 0.2463$ και $NDCG@10 = 0.6438$ δείχνουν ότι το collaborative μοντέλο δεν εντοπίζει απλώς περισσότερες relevant ταινίες, αλλά τις τοποθετεί και σε καλύτερες θέσεις μέσα στη λίστα. Η τιμή $HitRate@10 = 0.8703$ είναι επίσης ιδιαίτερα υψηλή, καθώς δείχνει ότι περίπου στο 87% των χρηστών υπάρχει τουλάχιστον μία relevant πρόταση μέσα στις πρώτες δέκα.

Η βελτίωση αυτή οφείλεται στο γεγονός ότι το ALS αξιοποιεί τα μοτίβα αξιολογήσεων όλων των χρηστών και όχι μόνο τα χαρακτηριστικά των ταινιών. Μέσω της matrix factorization, οι χρήστες και οι ταινίες αναπαρίστανται σε έναν κοινό λανθάνοντα χώρο, όπου το μοντέλο μπορεί να εντοπίσει σχέσεις που δεν είναι απαραίτητα εμφανείς από τα μεταδεδομένα. Για παράδειγμα, δύο ταινίες μπορεί να θεωρηθούν κοντινές όχι επειδή έχουν το ίδιο genre ή τον ίδιο σκηνοθέτη, αλλά επειδή αξιολογούνται θετικά από παρόμοιους χρήστες. Αυτό επιτρέπει στο collaborative μοντέλο να παράγει πιο στοχευμένες προτάσεις. Από την άλλη πλευρά, η Diversity του collaborative μοντέλου είναι 0.5875, τιμή που δείχνει ικανοποιητική αλλά πιο συγκεντρωμένη ποικιλία, καθώς οι προτάσεις βασίζονται περισσότερο σε κοινά μοτίβα προτίμησης.

Το hybrid μοντέλο παρουσιάζει την καλύτερη συνολική απόδοση. Συγκεκριμένα, πετυχαίνει $Precision@10 = 0.3123$, $Recall@10 = 0.2565$, $MAP@10 = 0.2781$, $NDCG@10 = 0.6713$ και $HitRate@10 = 0.8942$. Οι τιμές αυτές είναι υψηλότερες από τις αντίστοιχες τιμές τόσο του content-based όσο και του collaborative μοντέλου. Αυτό δείχνει ότι ο συνδυασμός collaborative και content-based πληροφορίας λειτουργεί θετικά, καθώς το μοντέλο αξιοποιεί τα συλλογικά μοτίβα αξιολόγησης του ALS, ενώ παράλληλα ενσωματώνει και την ομοιότητα περιεχομένου μέσω TF-IDF. Με άλλα λόγια, το hybrid μοντέλο δεν βασίζεται αποκλειστικά ούτε στη συμπεριφορά άλλων χρηστών ούτε στα περιγραφικά χαρακτηριστικά των ταινιών, αλλά συνδυάζει και τις δύο πηγές πληροφορίας.

Ιδιαίτερα σημαντική είναι και η τιμή $HitRate@10 = 0.8942$, η οποία δείχνει ότι σχεδόν στο 89,42% των χρηστών το hybrid μοντέλο κατάφερε να προτείνει τουλάχιστον μία relevant ταινία μέσα στις πρώτες δέκα προτάσεις. Αυτό αποτελεί ισχυρή ένδειξη πρακτικής αποτελεσματικότητας, καθώς δείχνει ότι το μοντέλο σπάνια αποτυγχάνει πλήρως να βρει κάτι σχετικό για τον χρήστη. Παράλληλα, η τιμή $NDCG@10 = 0.6713$ δείχνει ότι οι σχετικές ταινίες δεν εμφανίζονται απλώς στη λίστα, αλλά τείνουν να τοποθετούνται και σε υψηλές θέσεις.

Ένα ακόμη ενδιαφέρον αποτέλεσμα είναι η πολύ υψηλή Diversity του hybrid μοντέλου, με τιμή 0.9094. Στη συγκεκριμένη υλοποίηση, η Diversity του hybrid μοντέλου υπολογίζεται στον χώρο περιεχομένου, με βάση τα TF-IDF διανύσματα των προτεινόμενων ταινιών. Η υψηλή αυτή τιμή δείχνει ότι οι τελικές λίστες προτάσεων δεν περιορίζονται σε υπερβολικά όμοιες ταινίες, αλλά εμφανίζουν σημαντική εσωτερική διαφοροποίηση ως προς τα περιγραφικά χαρακτηριστικά τους. Επομένως, το hybrid μοντέλο καταφέρνει να βελτιώσει τις μετρικές συνάφειας και κατάταξης, χωρίς να μειώνει την ποικιλία των προτάσεων.

Συνολικά, η σύγκριση των τριών μοντέλων δείχνει ότι το content-based μοντέλο προσφέρει ικανοποιητική ποικιλία και ερμηνευσιμότητα, αλλά υστερεί ως προς τη συνάφεια των προτάσεων. Το collaborative μοντέλο ALS βελτιώνει σημαντικά την ακρίβεια και την ποιότητα κατάταξης, αξιοποιώντας τα μοτίβα αξιολογήσεων των χρηστών. Το hybrid μοντέλο εμφανίζει την καλύτερη συνολική συμπεριφορά, καθώς συνδυάζει τα πλεονεκτήματα και των δύο προσεγγίσεων και επιτυγχάνει τις υψηλότερες τιμές στις περισσότερες μετρικές αξιολόγησης.

5.2 Γενικά Συμπεράσματα και Μελλοντικές Επεκτάσεις

Η παρούσα εργασία είχε ως στόχο την ανάπτυξη και αξιολόγηση διαφορετικών προσεγγίσεων συστημάτων σύστασης ταινιών, χρησιμοποιώντας το MovieLens 32M dataset. Στο πλαίσιο αυτό υλοποιήθηκαν τρία μοντέλα: ένα content-based σύστημα, ένα collaborative filtering σύστημα βασισμένο σε ALS matrix factorization και ένα weighted hybrid σύστημα που συνδυάζει τις δύο προηγούμενες προσεγγίσεις. Η αξιολόγηση των μοντέλων πραγματοποιήθηκε με top-k μετρικές, οι οποίες είναι κατάλληλες για recommender systems που παράγουν λίστες προτάσεων και όχι απλώς προβλέψεις αριθμητικών βαθμολογιών.

Το πρώτο βασικό συμπέρασμα είναι ότι η χρήση περιγραφικών χαρακτηριστικών μπορεί να προσφέρει χρήσιμη πληροφορία για την παραγωγή προτάσεων, ειδικά όταν υπάρχει διαθέσιμο ιστορικό θετικών αξιολογήσεων για τον χρήστη. Το content-based μοντέλο αξιοποιεί χαρακτηριστικά όπως genres, tags και εμπλουτισμένα TMDb μεταδεδομένα, δημιουργώντας προτάσεις που βασίζονται στην ομοιότητα περιεχομένου με ταινίες που ο χρήστης έχει ήδη αξιολογήσει θετικά. Ωστόσο, η απόδοσή του στις μετρικές συνάφειας και κατάταξης είναι χαμηλότερη σε σχέση με τα υπόλοιπα δύο μοντέλα, γεγονός που δείχνει ότι η πληροφορία περιεχομένου, αν και σημαντική, δεν αρκεί από μόνη της για να αποτυπώσει πλήρως τις προτιμήσεις των χρηστών.

Το δεύτερο συμπέρασμα είναι ότι το collaborative filtering και ειδικότερα το ALS αξιοποιεί πιο αποτελεσματικά τα μοτίβα αξιολογήσεων των χρηστών. Μέσα από τη μήτρα αλληλεπιδράσεων χρήστη-ταινίας, το μοντέλο μαθαίνει λανθάνουσες σχέσεις ανάμεσα σε χρήστες και αντικείμενα, οι οποίες δεν αποτυπώνονται απαραίτητα στα εμφανή χαρακτηριστικά των ταινιών. Αυτό οδηγεί σε σημαντική βελτίωση της συνάφειας και της κατάταξης των προτάσεων, καθώς το σύστημα μπορεί να αξιοποιήσει τη συλλογική συμπεριφορά των χρηστών και να εντοπίσει ταινίες που είναι πιθανό να ενδιαφέρουν έναν χρήστη, ακόμη και αν δεν είναι προφανώς όμοιες με βάση τα περιγραφικά τους χαρακτηριστικά.

Το τρίτο και σημαντικότερο συμπέρασμα είναι ότι η υβριδική προσέγγιση προσφέρει την καλύτερη συνολική απόδοση μεταξύ των τριών μοντέλων που εξετάστηκαν. Το hybrid μοντέλο συνδυάζει το collaborative score του ALS με το content-based score που προκύπτει από την ομοιότητα του προφίλ χρήστη με τα TF-IDF διανύσματα των ταινιών. Η χρήση του βάρους $\alpha = 0.70$ δίνει μεγαλύτερη έμφαση στο collaborative component, το οποίο

αποδείχθηκε ιδιαίτερα αποτελεσματικό στις μετρικές συνάφειας, ενώ το content-based component λειτουργεί συμπληρωματικά, προσθέτοντας πληροφορία περιεχομένου. Το αποτέλεσμα είναι ένα μοντέλο που επιτυγχάνει καλύτερη ισορροπία ανάμεσα στην ακρίβεια, την ποιότητα κατάταξης και την εσωτερική ποικιλία των προτάσεων.

Ιδιαίτερα σημαντικό είναι ότι το hybrid μοντέλο πετυχαίνει τις υψηλότερες τιμές στις βασικές μετρικές αξιολόγησης, όπως Precision@10, Recall@10, MAP@10, NDCG@10 και HitRate@10. Αυτό δείχνει ότι ο συνδυασμός collaborative και content-based πληροφορίας λειτουργεί θετικά και οδηγεί σε πιο αποτελεσματικές λίστες προτάσεων. Παράλληλα, η υψηλή τιμή της Diversity δείχνει ότι οι τελικές προτάσεις δεν περιορίζονται σε υπερβολικά όμοιες ταινίες, αλλά διατηρούν σημαντική εσωτερική διαφοροποίηση ως προς το περιεχόμενο.

Η αξιολόγηση της εργασίας πραγματοποιήθηκε σε offline περιβάλλον, με βάση ιστορικά ratings. Η επιλογή αυτή είναι κατάλληλη για την πειραματική σύγκριση διαφορετικών μοντέλων, καθώς επιτρέπει την εφαρμογή κοινών μετρικών και κοινών συνθηκών αξιολόγησης. Παράλληλα, πρέπει να σημειωθεί ότι η offline αξιολόγηση αποτυπώνει τη συμπεριφορά των μοντέλων πάνω σε υπάρχοντα δεδομένα και δεν ταυτίζεται απαραίτητα με την πραγματική εμπειρία ενός χρήστη σε ένα ζωντανό σύστημα σύστασης. Επομένως, τα αποτελέσματα παρέχουν μια αξιόπιστη πειραματική ένδειξη της απόδοσης των μοντέλων, ενώ η αξιολόγηση σε πραγματικό περιβάλλον χρήσης θα μπορούσε να αποτελέσει αντικείμενο μελλοντικής διερεύνησης.

Κατά τη διάρκεια της ανάπτυξης πραγματοποιήθηκαν πολλαπλές δοκιμές και παραλλαγές και στις τρεις προσεγγίσεις που εξετάστηκαν. Στο content-based μοντέλο δοκιμάστηκαν διαφορετικοί τρόποι κατασκευής του περιεχομένου των ταινιών, διαφορετικά βάρη στα επιμέρους πεδία, διαφορετικές ρυθμίσεις TF-IDF και διαφορετικοί τρόποι δημιουργίας του προφίλ χρήστη. Αντίστοιχα, στο collaborative μοντέλο εξετάστηκαν παραλλαγές που σχετίζονται με την παραμετροποίηση του ALS, ενώ στο hybrid μοντέλο δοκιμάστηκαν διαφορετικοί τρόποι συνδυασμού και διαφορετικές τιμές της παραμέτρου α . Η τελική μορφή των μοντέλων που παρουσιάστηκε στην εργασία δεν επιλέχθηκε τυχαία, αλλά αποτέλεσε το αποτέλεσμα αυτών των πειραματισμών, με κριτήριο τη συνολική απόδοση, τη σταθερότητα των αποτελεσμάτων και την ερμηνευσιμότητα της μεθόδου.

Το γεγονός ότι πιο σύνθετες ή διαφορετικές παραλλαγές δεν οδήγησαν απαραίτητα σε καλύτερες μετρικές δείχνει ότι η αύξηση της πολυπλοκότητας ενός συστήματος σύστασης δεν συνεπάγεται πάντα αντίστοιχη βελτίωση της απόδοσης. Στο συγκεκριμένο dataset και στο

συγκεκριμένο evaluation setup, η τελική υβριδική προσέγγιση με σταθμισμένο συνδυασμό collaborative και content-based score παρουσίασε την καλύτερη ισορροπία ανάμεσα στη συνάφεια, την ποιότητα κατάταξης και την ποικιλία των προτάσεων. Επομένως, η τελική επιλογή του μοντέλου βασίστηκε όχι μόνο στη θεωρητική ισχύ της μεθόδου, αλλά και στην εμπειρική της συμπεριφορά κατά την αξιολόγηση.

Ως μελλοντική επέκταση, μια ουσιαστική κατεύθυνση θα ήταν η δυναμική προσαρμογή του βάρους ανά χρήστη. Στην παρούσα εργασία χρησιμοποιήθηκε μία σταθερή τιμή $\alpha = 0.70$, η οποία εφαρμόστηκε σε όλους τους χρήστες. Ωστόσο, διαφορετικοί χρήστες ενδέχεται να ωφελούνται από διαφορετική ισορροπία ανάμεσα στο collaborative και στο content-based component. Για παράδειγμα, ένας χρήστης με πλούσιο ιστορικό αξιολογήσεων μπορεί να υποστηρίζεται καλύτερα από το collaborative μέρος, ενώ ένας χρήστης με περιορισμένο ιστορικό μπορεί να επωφελείται περισσότερο από την περιγραφική πληροφορία των ταινιών. Μια τέτοια προσαρμογή θα μπορούσε να οδηγήσει σε πιο εξατομικευμένο υβριδικό συνδυασμό.

Επιπλέον, θα μπορούσε να εξεταστεί μια πιο εξειδικευμένη αξιολόγηση ανά κατηγορία χρηστών. Στην παρούσα εργασία τα αποτελέσματα παρουσιάζονται ως μέσοι όροι στο σύνολο των αξιολογημένων χρηστών. Μια μελλοντική ανάλυση θα μπορούσε να διαχωρίσει τους χρήστες με βάση το πλήθος των αξιολογήσεών τους, τη δραστηριότητά τους ή τη θεματική ποικιλία των προτιμήσεών τους. Με αυτόν τον τρόπο θα μπορούσε να εξεταστεί αν το κάθε μοντέλο αποδίδει διαφορετικά σε χρήστες με πλούσιο ιστορικό, σε χρήστες με περιορισμένο ιστορικό ή σε χρήστες με πιο εξειδικευμένες προτιμήσεις.

Συμπερασματικά, η εργασία έδειξε ότι ο συνδυασμός collaborative και content-based πληροφορίας μπορεί να βελτιώσει ουσιαστικά την απόδοση ενός συστήματος σύστασης ταινιών. Το content-based μοντέλο αξιοποίησε την περιγραφική πληροφορία των ταινιών, το collaborative μοντέλο αξιοποίησε τα συλλογικά μοτίβα αξιολογήσεων, ενώ το hybrid μοντέλο συνδύασε τις δύο προσεγγίσεις και πέτυχε την καλύτερη συνολική απόδοση. Τα αποτελέσματα δείχνουν ότι η υβριδική προσέγγιση αποτελεί την πιο αποτελεσματική λύση στο πλαίσιο της συγκεκριμένης εργασίας, καθώς προσφέρει καλύτερη ισορροπία ανάμεσα στη συνάφεια, την ποιότητα κατάταξης και την ποικιλία των προτάσεων.

Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

- Da Poian, V., Theiling, B., Clough, L., McKinney, B., Major, J., Chen, J., & Hörst, S. (2023). Exploratory data analysis (EDA) machine learning approaches for ocean world analog mass spectrometry. *Frontiers in Astronomy and Space Sciences*, 10. <https://doi.org/10.3389/fspas.2023.1134141>
- Bobadilla, J., Ortega, F., Hernando, A., & Bernal, J. (2012). A collaborative filtering approach to mitigate the new user cold start problem. *Knowledge-Based Systems*, 26, 225–238. <https://doi.org/10.1016/j.knosys.2011.07.021>
- García, S., Luengo, J., & Herrera, F. (n.d.). *Intelligent Systems Reference Library 72 Data Preprocessing in Data Mining*. <http://www.springer.com/series/8578>
- Goyani, M., & Chaurasiya, N. (2020). A Review of Movie Recommendation System: Limitations, Survey and Challenges. *Electronic Letters on Computer Vision and Image Analysis*, 19(3), 18–37. <https://doi.org/10.5565/rev/elcvia.1232>
- Isinkaye, F. O., Folajimi, Y. O., & Ojokoh, B. A. (2015). Recommendation systems: Principles, methods and evaluation. In *Egyptian Informatics Journal* (Vol. 16, Issue 3, pp. 261–273). Elsevier B.V. <https://doi.org/10.1016/j.eij.2015.06.005>
- Jayalakshmi, S., Ganesh, N., Čep, R., & Murugan, J. S. (2022). Movie Recommender Systems: Concepts, Methods, Challenges, and Future Directions. *Sensors*, 22(13). <https://doi.org/10.3390/s22134904>
- Kamble, K., Bhadarge, P., Bhakte, S., Kulkarni, O., Kadam, R., & Pinjarkar, L. (2025). Movie Recommendation System Using Hybrid Based Method. *2025 International Conference on Automation and Computation, AUTOCOM 2025*, 119–126. <https://doi.org/10.1109/AUTOCOM64127.2025.10956922>
- Recommender Systems Handbook. (2011). In *Recommender Systems Handbook*. Springer US. <https://doi.org/10.1007/978-0-387-85820-3>
- Roy, D., & Dutta, M. (2022). A systematic review and research perspective on recommender systems. *Journal of Big Data*, 9(1). <https://doi.org/10.1186/s40537-022-00592-5>

- Fkih, F. (2022). Similarity measures for Collaborative Filtering-based Recommender Systems: Review and experimental comparison. In *Journal of King Saud University - Computer and Information Sciences* (Vol. 34, Issue 9, pp. 7645–7669). King Saud bin Abdulaziz University. <https://doi.org/10.1016/j.jksuci.2021.09.014>
- Gazdar, A., & Hidri, L. (2020). *A new Similarity Measure for Collaborative Filtering based Recommender Systems*. <https://doi.org/10.1016/j.knosys.2019.105058>
- Hu, Y., Volinsky, C., & Koren, Y. (2008). Collaborative filtering for implicit feedback datasets. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 263–272. <https://doi.org/10.1109/ICDM.2008.22>
- Martins, G. B., Papa, J. P., & Adeli, H. (2020). Deep learning techniques for recommender systems based on collaborative filtering. In *Expert Systems* (Vol. 37, Issue 6). Blackwell Publishing Ltd. <https://doi.org/10.1111/exsy.12647>
- Papadakis, H., Papagrigoriou, A., Panagiotakis, C., Kosmas, E., & Fragopoulou, P. (2022). Collaborative filtering recommender systems taxonomy. *Knowledge and Information Systems*, 64(1), 35–74. <https://doi.org/10.1007/s10115-021-01628-7>
- Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating Collaborative Filtering Recommender Systems, Volume 22, Issue 1. ACM Transactions on Information Systems. <https://doi.org/10.1145/963770.963772>
- Jadon, A., & Patil, A. (2024). A Comprehensive Survey of Evaluation Techniques for Recommendation Systems. <https://doi.org/10.13140/RG.2.2.29818.30401>
- Mobasher, Bamshad. (2011). Proceedings of the fifth ACM conference on Recommender systems. Vargas, S., & Castells, P. . Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems. ACM. <https://doi.org/10.1145/2043932.2043955>
- Tamm, Y. M., Damdinov, R., & Vasilev, A. (2021). Quality metrics in recommender systems: Do We calculate metrics consistently? *RecSys 2021 - 15th ACM Conference on Recommender Systems*, 708–713. <https://doi.org/10.1145/3460231.3478848>
- Adomavicius, G., & Tuzhilin, A. (2005). *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions*. <https://doi.org/10.1109/TKDE.2005.99>

- Burke, Robin. (2002). Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction. 12. <https://doi.org/10.1023/A:1021240730564>
- Cremonesi, Paolo & Turrin, Roberto & Airoidi, Fabio. (2011). Hybrid algorithms for recommending new items in personal TV. CEUR Workshop Proceedings. 720. <https://doi.org/10.1145/2039320.2039325>
- Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. Science, 349(6245), 255–260. <https://doi.org/10.1126/science.aaa8415>
- Sarker, I. H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. SN Computer Science, 2, 160. <https://doi.org/10.1007/s42979-021-00592-x>
- García, S., Luengo, J., & Herrera, F. (2015). Data Preprocessing in Data Mining. Springer. <https://doi.org/10.1007/978-3-319-10247-4>
- Morgenthaler, S. (2009). Exploratory data analysis. Wiley Interdisciplinary Reviews: Computational Statistics, 1(1), 33–44. <https://doi.org/10.1002/wics.2>
- GeeksforGeeks. (2025, December 27). What are Recommender Systems? GeeksforGeeks. <https://www.geeksforgeeks.org/machine-learning/what-are-recommender-systems/>
- NVIDIA. (n.d.). Recommendation system. NVIDIA Glossary. Retrieved June 29, 2026, from <https://www.nvidia.com/en-us/glossary/recommendation-system/>
- Takyar, A. (n.d.). Exploratory Data Analysis: Overview, techniques, tools and applications. LeewayHertz. Retrieved June 29, 2026, from <https://www.leewayhertz.com/what-is-exploratory-data-analysis/>
- Chanaka. (2024, June 15). Data Preprocessing in Machine Learning. Medium. <https://medium.com/@ChanakaDev/data-preprocessing-in-machine-learning-940f4769a95a>
- Evidently AI Team. (2025, February 14). 10 metrics to evaluate recommender and ranking systems. Evidently AI. <https://www.evidentlyai.com/ranking-metrics/evaluating-recommender-systems/>

- Vutukuri, K. (2025, November 3). 34. Hybrid Recommender Systems: The Best of Both Worlds. Medium. <https://medium.com/@kiranvutukuri/34-hybrid-recommender-systems-the-best-of-both-worlds-a658d0ed44de>
- F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. <https://doi.org/10.1145/2827872>

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.